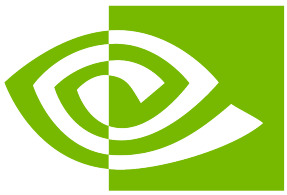


NVIDIA Augmented Reality (AR) SDK User Guide

Release 1.2.0

NVIDIA Corporation

Sep 10, 2026

Contents

Download this guide as a PDF

The NVIDIA Augmented Reality SDK (AR SDK) is a comprehensive collection of AI-powered features for real-time modeling and tracking of human faces and bodies from video.

The AR SDK enables developers to build state-of-the-art video processing applications with AI-powered features, such as body pose estimation, face detection, landmark tracking, gaze redirection, and facial expressions. The SDK is powered by NVIDIA graphics processing units (GPUs) with Tensor Cores, supporting high throughput and low latency processing.

The AR SDK has the following features:

- **Face detection and tracking** detects, localizes, and tracks human faces in images or videos by using bounding boxes.
- **Facial landmark detection and tracking** predicts and tracks the pixel locations of human facial landmark points and head poses in images or videos. It can predict 68 and 126 landmark points. The 68 detected facial landmarks follow the Multi-PIE 68 point mark-up information in Facial point annotations. The 126 facial landmark points detector can predict more points on the cheeks, the eyes, and on laugh lines.
- **Legacy 3D Body Pose Tracking** predicts and tracks 34 body keypoints in 2D and 3D from images or videos. It outputs joint rotations and supports multiple people in full-body and upper-body images or videos. For new applications, use *3D Body Pose Estimation*. Refer to *Appendix B: 3D Body Pose Keypoint Format* for keypoint details.
- **3D Body Pose Estimation** estimates body pose from video frames by using caller-provided tracking boxes. It predicts 77 body keypoints in 2D and in 3D camera coordinates, and it can output rest pose, root pose, and joint rotations. Refer to *Appendix B: 3D Body Pose Keypoint Format* for the keypoint format.
- **Eye Contact** estimates the gaze angles of a person in an image or video and redirects the gaze to make it frontal. It can operate in two modes. One mode where head pose and gaze angles are estimated in camera coordinates without any redirection and another where, in addition to estimation, the eyes of the person are redirected to make eye contact with the camera within a permissible range.
- **Facial Expression Estimation** estimates face expression coefficients from the provided facial landmarks.
- **LipSync** animates a person's video using an audio input by animating the lip motion to match that of the audio.
- **Active Speaker Detection** identifies which person in a video is currently speaking by analyzing both video frames and synchronized audio tracks. It supports multiple audio tracks and outputs face bounding boxes with tracking IDs and speaking status.

Note

The Windows SDK supports x64 systems and Windows on Arm (ARM64) systems with a supported iGPU such as RTX Spark. On Windows on Arm, LipSync and Active Speaker Detection are not supported, and only the Body Pose Estimation / Body Detection path is available.

Chapter 1. Get Started on Windows

The AR SDK requires specific NVIDIA GPUs, a specific version of the Windows OS, and other associated software on which the SDK depends. An external monitor is also required.

This SDK is designed and optimized for client-side application integration and local deployment. We do not officially support the testing, experimentation, deployment of this SDK to a datacenter/cloud environment.

1.1. Hardware Requirements

The AR SDK is supported on NVIDIA GPUs with Tensor Cores on the following Windows architectures:

- x64 systems with supported discrete NVIDIA GPUs based on the NVIDIA Turing™, NVIDIA Ampere™, NVIDIA Ada™, or NVIDIA Blackwell™ architecture.
- Windows on Arm (ARM64) systems with a supported iGPU such as RTX Spark.

1.2. Software Requirements

The AR SDK requires a specific version of the Windows OS and other associated software on which the SDK depends. The NVIDIA CUDA® and inference runtime dependencies are bundled with the SDK: TensorRT™ for x64 and TensorRT-RTX for ARM64. Refer to *Install the AR SDK*.

Software	Required Version
Windows OS	x64: Windows 10 or Windows 11 ARM64: Windows 11 (Windows on Arm, such as RTX Spark)
NVIDIA Graphics Driver for Windows	x64: 570.65 or later ARM64: 615 or later (Windows on Arm, such as RTX Spark)

1.2.1. Install the AR SDK

The AR SDK comprises the SDK Core and a set of optional features that can be downloaded and installed individually. The SDK Core and features are distributed through the NVIDIA GPU Cloud (NGC) platform.

The SDK Core includes the API headers, library files, and runtime dependencies. It does not include the libraries and models that are required to run any of the features. After installation, the SDK Core provides a script to fetch and install features from NGC.

To download the AR SDK Core, navigate to NVIDIA Augmented Reality SDK Collection on the NVIDIA GPU Cloud (NGC) website, and follow the instructions in the "Getting Started" section.

The SDK is delivered as a compressed `.zip` file.

- To install the SDK Core, extract the contents of the AR SDK Core package to your desired location.
- To use the `install_feature.ps1` script for feature installation, obtain your NGC API key by using the steps in Generating a Personal API Key in the NGC User Guide. Then use the following template commands to install a feature:

```
# Allow download script to run without any restrictions or warnings:
Set-ExecutionPolicy Bypass -Scope Process

cd /path/to/ARSDK/features/
$Env:NGC_CLI_API_KEY = "<your-ngc-api-key>"
./install_feature.ps1 -gpu <your_GPU> -features <feature_name> -ngc_org
↪<NGC_ORG> -ngc_team <NGC_TEAM>
```

The following lists the arguments and usage for the `install_feature.ps1` script:

```
========================================
AR SDK Feature Installation Script
========================================

Usage: .\install_feature.ps1 [OPTIONS]

Options:
  -gpu <gpu>            : GPU compute capability (for example, 75, 86, 89,
↪120, 121)
                          Optional - will auto-detect if not specified
  -features <features>  : Feature(s) to install (case-insensitive, default:
↪all)
                          Can be a single feature, comma-separated list, or
↪'all'
  -version <version>    : Feature version (default: latest from NGC)
                          applicable when specific features are specified
↪(not 'all')
  -ngc_org <org>        : NGC organization (default: nvidia)
  -ngc_team <team>      : NGC team (default: maxine)
  -list_features        : Query NGC for complete list of available features
  -help                 : Show this help message

Examples:
  # Install single feature (auto-detect GPU and latest compatible version):
  .\install_feature.ps1 -features nvargazeredirection
```

(continues on next page)

(continued from previous page)

```
  # Install single feature with specific GPU:
  .\install_feature.ps1 -gpu 89 -features nvargazeredirection

  # Install single feature with specific version:
  .\install_feature.ps1 -features nvargazeredirection -version 1.1.0.0

  # Install multiple features (comma-separated):
  .\install_feature.ps1 -gpu 89 -features nvargazeredirection,nvarlipsync,
↪nvarbodydetection

  # Install all features:
  .\install_feature.ps1 -gpu 89 -features all

  # Install all features on Windows on Arm (such as RTX Spark):
  .\install_feature.ps1 -gpu 121 -features all

Example AR SDK features (non-exhaustive, case-insensitive):
  * nvarbodyposeestimation  - Body pose estimation
  * nvarbodydetection       - Body detection
  * nvarlandmarkdetection   - Face landmark Detection
  * nvargazeredirection     - Eye contact/gaze redirection
  * nvarlipsync             - Lip synchronization
  * nvarfaceexpressions     - Face expression estimation

For complete list of features, use: -list_features

Supported GPU compute capabilities: 75, 86, 89, 120, 121 (Windows on Arm)
```

On Windows on Arm (for example, RTX Spark), specify `-gpu 121` (or the `rtx-spark` alias). The script installs the Windows on Arm (`woa`) feature libraries and models. If you omit `-gpu`, the script attempts to auto-detect the GPU on the target machine.

For a full list of NVIDIA GPUs and their corresponding CUDA Compute Capability versions, see the CUDA GPU Compute Capability Table.

1.2.2. Sample Applications

To build and run the sample applications, fetch the open source repository from https://github.com/NVIDIA-Maxine/AR-SDK-Samples and follow the instructions in `README.md`.

1.2.3. Environment Variables

The AR SDK uses the following environment variables:

- `NVAR_MODEL_DIR`

  If the application does not provide the path to the model's directory by setting the `NvAR_Parameter_Config_ModelDir` string, the SDK tries to load the models from the path in the `NVAR_MODEL_DIR` environment variable.

  The SDK installer sets `NVAR_MODEL_DIR` to `%ProgramFiles%\NVIDIA Corporation\NVIDIA AR SDK\models`.

- `NV_AR_SDK_PATH`

  By default, applications that use the SDK try to load the SDK DLL and its dependencies from the SDK installation directory, such as `%ProgramFiles%\NVIDIA Corporation\NVIDIA AR SDK\models`. Applications might also want to include and load the SDK DLL and its dependencies directly from the application folder.

  To prevent the files from being loaded from the installation directory, the application can set this environment variable to `USE_APP_PATH`. If `NV_AR_SDK_PATH` is set to `USE_APP_PATH`, instead of loading the binaries from the Program Files installation directory, the SDK follows the standard OS search order to load the binaries, such as the app folder followed by the `PATH` environment variable.

  Important

  Set this variable only for the application process. Setting this variable as a user or system variable affects other applications that use the SDK.

Both variables accept Unicode (non-ASCII) paths. Refer to *Unicode Path Support*.

1.2.4. Unicode Path Support

The AR SDK supports Unicode (non-ASCII) characters in all file and directory paths on Windows, including the following:

- Model directory paths that you set with the `NvAR_Parameter_Config_ModelDir` string through `NvAR_SetString`, or with the `NVAR_MODEL_DIR` environment variable.
- DLL load paths, such as the AR SDK installation directory.
- Log file paths.

The SDK resolves paths that contain Chinese, Japanese, Korean, accented Latin, or other non-ASCII characters, such as the following path:

```
C:\  \  \NVIDIA_AR_SDK\models
```

All path string parameters accept strings encoded in UTF-8 on both Windows and Linux.

Note

On Linux, the operating system handles UTF-8 paths natively. The Unicode path improvements apply mainly to Windows, where the SDK previously relied on the system ANSI code page.

1.2.4.1 Command-Line Arguments in Sample Applications

On Windows, the AR SDK sample applications use wide-character entry points (`wmain`, or `GetCommandLineW` with `CommandLineToArgvW`) to receive command-line arguments. The samples convert those arguments to UTF-8 before passing them to the SDK.

1.2.5. Performance Reference

This table lists performance data for features of the NVIDIA AR SDK on a few supported GPU architectures.

	Latency (in milliseconds)			
Feature	**RTX 2060**	**RTX 3080**	**RTX 4090**	**RTX 5090**
Face Detection and Tracking	0.414	0.312	0.207	0.389
Facial Landmark Detection and Tracking	0.93	0.783	0.526	0.869
3D Body Pose Tracking (Legacy)	3.67	2.56	1.27	2.86
Eye Contact	4.11	2.36	1.91	2.32
Facial Expression Estimation	1.54	0.981	0.674	1.07
LipSync	71.9	34.7	27.6	20.8
Active Speaker Detection	16.5	9.46	7.28	8.67
3D Body Pose Estimation	106	49.8	21.9	19.2

Chapter 2. Get Started on Linux

The SDK requires specific NVIDIA GPUs, a specific distribution of Linux, and other associated software on which the SDK depends.

This SDK is designed and optimized for server-side (datacenter/cloud) deployment. We do not officially support the testing, experimentation, and production deployment of this SDK to client-side application integration and local deployment.

2.1. Hardware Requirements

The AR SDK is compatible with GPUs that are based on NVIDIA Turing™, NVIDIA Ampere™, NVIDIA Ada™, NVIDIA Hopper™, or NVIDIA Blackwell™ architecture.

Note

For best performance with NVIDIA T4 and other server GPUs, make sure that you use a server that meets the thermal and airflow requirements for these types of products. For the latest list of qualified servers, refer to the Qualified System Catalog.

2.2. Software Requirements

The SDK requires a specific version of Linux and other associated software on which the SDK depends.

Software	Required Version
Linux	Ubuntu 20.04, 22.04, 24.04 Debian 11, 12 Rocky Linux 8 or 9 RHEL 8 or 9
NVIDIA Graphics Driver for Linux	570.26 or later

2.2.1. Install the AR SDK

The AR SDK comprises the SDK Core and a set of optional features that can be downloaded and installed individually. The SDK Core and features are distributed through the NVIDIA GPU Cloud (NGC) platform.

The SDK Core includes the API headers, library files, and runtime dependencies. It does not include the libraries and models that are required to run any of the features. After installation, the SDK Core provides a script to fetch and install features from NGC.

To download the AR SDK Core, navigate to NVIDIA Augmented Reality SDK Collection on the NVIDIA GPU Cloud (NGC) website, and follow the instructions in the "Getting Started" section.

The SDK is delivered as a `.tgz` package, which is a compressed tar format. Before extracting the SDK Core package, install `xz-utils` if it is not already installed. For example, on Ubuntu use the command `sudo apt install xz-utils`.

- To install the SDK Core, extract the contents of the AR SDK Core package as follows:

```
sudo tar -xvf ARSDK_linux_<version>.tgz -C /usr/local
```

- To use the `install_feature.sh` script for feature installation, obtain your NGC API key by using the steps in Generating a Personal API Key in the NGC User Guide. Then use the following template commands to install a feature:

```
cd /usr/local/ARSDK/features
export NGC_CLI_API_KEY=<your_API_key>
sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh --gpu <your_GPU>
↪--feature <feature_name> --ngc-org <NGC_ORG> --ngc-team <NGC_TEAM>
```

Note

When running in privileged mode, omit `sudo --preserve-env=NGC_CLI_API_KEY`; however, the `NGC_CLI_API_KEY` environment variable must still be set.

The following lists the arguments and usage for the `install_feature.sh` script:

```
=======================================
ARSDK Feature Installation Script
=======================================
Usage: install_feature.sh [OPTIONS]
Options:
  -g, --gpu <gpu>        GPU architecture (e.g., a100, b200)
  -f, --feature <list>   Feature(s) to install (case-insensitive, default:
↪all)
                         Can be a single feature, comma-separated list, or
↪'all'
  -v, --version <ver>    Feature version (default: latest from NGC)
                         applicable when specific features are specified (not
↪'all')
  --ngc-org <org>        NGC organization (default: nvidia)
  --ngc-team <team>      NGC team (default: maxine)
  --list-features        Query NGC for complete list of available features
  -h, --help             Show this help message
```

(continues on next page)

(continued from previous page)

```
Examples:
  # Install single feature (auto-detect GPU and latest compatible version):
  sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh -f
↪nvargazeredirection

  # Install single feature with specific GPU:
  sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh -f
↪nvargazeredirection -g a100

  # Install single feature with specific version:
  sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh -f
↪nvargazeredirection -v 1.1.0.0

  # Install multiple features (comma-separated):
  sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh -f
↪nvargazeredirection,nvarlipsync,nvarbodydetection

  # Install all features:
  sudo --preserve-env=NGC_CLI_API_KEY ./install_feature.sh -f all

Example AR SDK features (non-exhaustive, case-insensitive):
  • nvarbodyposeestimation  - Body pose estimation
  • nvarbodydetection       - Body detection
  • nvarlandmarkdetection   - Face landmark tracking
  • nvargazeredirection     - Eye contact/gaze redirection
  • nvarlipsync             - Lip synchronization
  • nvarfaceexpressions     - Face expression estimation

For complete list of features, use: --list-features

Note: Dependencies are installed automatically and won't be reinstalled if
↪already present

Supported GPUs: a40/l40/l4/a30/b200/a2/h100/a10/t4/b100/a16/a100/b40
```

The following table lists supported GPUs and their corresponding CUDA Compute Capability versions:

GPU	CUDA CC
t4	7.5
a100, a30	8.0
a2, a10, a16, a40	8.6
l4, l40	8.9
h100	9.0
b100, b200	10.0
b40	12.0

For a full list of NVIDIA GPUs and their corresponding CUDA Compute Capability versions, see the CUDA GPU Compute Capability Table.

2.2.2. Sample Applications

To build and run the sample applications, fetch the open source repository from https://github.com/NVIDIA-Maxine/AR-SDK-Samples and follow the instructions in `README.md`.

2.2.3. Performance Reference

This table lists performance data for features of the NVIDIA AR SDK on a few supported GPU architectures.

	Latency (in milliseconds)				Throughput (30 FPS)			
Feature	**T4**	**A40**	**L40**	**B40**	**T4**	**A40**	**L40**	**B40**
Face Detection and Tracking	0.42	0.26	0.19	0.241	79	128	175	138
Facial Landmark Detection and Tracking	0.91	0.61	0.44	0.549	36	54	75	60
3D Body Pose Tracking (Legacy)	9.4	3.89	2.34	2.88	3	8	14	11
Eye Contact	4.15	2.8	2.08	1.7	8	11	16	19
Facial Expression Estimation	1.49	0.85	0.53	0.63	22	39	62	52
LipSync	87.5	32.6	15.4	14.3	0	1	2	2
Active Speaker Detection	19.1	8.63	5.94	7.85	1	3	5	4
3D Body Pose Estimation	N/A	42.2	26.6	15.8	N/A	0	1	2

Note

Throughput is defined as maximum number of streams to achieve 30 FPS real-time performance.

2.2.4. Run SDK in a Container

The AR SDK can run in an isolated container. It is easier to try the sample apps without worrying about dependencies. And the docker image can be shared easily. Two more software packages are required: Docker Engine and NVIDIA Container Toolkit.

2.2.4.1 Install the Prerequisite Software

2.2.4.1.1 Docker Engine

Docker Engine is an open source containerization technology for building and containerizing your applications.

Please follow this link (https://docs.docker.com/engine/install) to install Docker Engine on your Linux distribution.

Please test Docker Engine on officially supported Linux distributions, such as Ubuntu 20.04.

2.2.4.1.2 NVIDIA Container Toolkit

The NVIDIA Container Toolkit enables users to build and run GPU-accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPUs.

Please follow this link (https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html) to install and configure NVIDIA Container Toolkit.

2.2.4.2 Fetch Sample Apps Repository

Fetch the open source repository from https://github.com/NVIDIA-Maxine/AR-SDK-Samples to a local directory `ar-sdk-samples`, which will be mounted when launching the container.

2.2.4.3 Build Docker Image

Download and install the SDK and all models as described in *Install the AR SDK*. Then proceed to `ar-sdk-samples/docker` and execute the `build_docker_image.sh` script. The building script pulls the base image (Ubuntu 20.04) from Docker Hub and installs required third-party libraries for running sample applications.

Once finished, you will see a Docker image `maxine_arsdk` created on your system using the `sudo docker image ls` command.

The repository name and the TAG can be customized by modifying the `ar-sdk-samples/docker/docker_config.sh` and rebuilding the Docker image.

2.2.4.4 Launch Container

Launch the container by executing the script `ar-sdk-samples/docker/run_docker_image.sh`. You can choose a directory on the host machine mapped to `/host` in the container by specifying `-m <host_dir>` for file sharing purposes.

To run sample apps, the `arsdk-samples` directory must exist under `<host_dir>`.

Once the container is launched, you are ready to try the ARSDK sample applications in `/host/arsdk-samples`. Only offline mode is supported inside the container. You can dump the output to the `/host` directory so that you can evaluate it in the host system for playback.

For Linux-specific sample app instructions, refer to `README.md` in the sample app repo.

Chapter 3. AR SDK API Architecture

Use the NVIDIA® AR SDK to enable an application to use the face tracking, facial landmark tracking, 3D Body Pose tracking, Eye Contact, and other features of the SDK.

3.1. Working with Features

3.1.1. Creating an Instance of a Feature Type

The *feature type* is a predefined structure that is used to access the SDK features. Each feature requires an instantiation of the feature type. Creating an instance of a feature type provides access to configuration parameters that are used when loading an instance of the feature type and to the input and output parameters that are provided at runtime when instances of the feature type are run.

1. Allocate memory for an *NvAR_FeatureHandle* structure.

   ```
   NvAR_FeatureHandle faceDetectHandle{};
   ```

2. Call the *NvAR_Create* function.

   In the call to the function, pass the following information:

   - A value of the `NvAR_FeatureID` enumeration to identify the feature type.
   - A pointer to the variable that you declared to allocate memory for an *NvAR_FeatureHandle* structure.

3. To create an instance of the face detection feature type, first include the feature header file at the top of your source file:

   ```
   #include "nvARFaceBoxDetection.h"
   ```

   To create the feature instance, use the following call:

   ```
   NvAR_Create(NvAR_Feature_FaceBoxDetection, &faceDetectHandle)
   ```

   This function creates a handle to the feature instance, which is required in function calls to get and set the properties of the instance and to load, run, or destroy the instance.

   Each `NvAR_Feature_<feature-name>` identifier is defined under the respective feature header; for example, `nvARFaceBoxDetection.h`. Ensure that the requested feature is installed so that the SDK has access to both feature libraries and required model files.

   During *NvAR_Create()*, the SDK loads the feature library from the location `<ar-sdk>/features/<feature>/<bin/lib>`. Without a feature installation, the call to this API function returns an error code.

   For instructions on how to install features, refer to the installation guide.

3.1.2. Getting and Setting Properties for a Feature Type

To prepare to load and run an instance of a feature type, you need to set the properties that the instance requires.

Here are some of the properties:

- The configuration properties that are required to load the feature type.
- Input and output properties that are provided when instances of the feature type are run.

For a complete list of properties, refer to *Key Values in the Properties of a Feature Type*.

To set properties, the AR SDK provides type-safe set accessor functions. If you need the value of a property that has been set by a set accessor function, use the corresponding get accessor function. For a complete list of set and get functions, refer to *Summary of AR SDK Accessor Functions*.

3.1.3. Setting Up the CUDA Stream

Some SDK features require a CUDA stream in which to run. For more information, refer to the NVIDIA CUDA Toolkit Documentation.

1. Initialize a CUDA stream by calling one of the following functions:
   - The CUDA Runtime API function `cudaStreamCreate()`.
   - *NvAR_CudaStreamCreate*.

     You can use the second function to avoid linking with the NVIDIA CUDA Toolkit libraries.
2. Call the *NvAR_SetCudaStream* function and provide the following information as parameters:
   - The created filter handle.

     For more information, refer to *Creating an Instance of a Feature Type*.
   - The key value NvAR_Parameter_Config(CUDAStream).

     For more information, refer to *Key Values in the Properties of a Feature Type*.
   - The CUDA stream that you created in the previous step.

The following example sets up a CUDA stream that was created by calling the *NvAR_CudaStreamCreate* function:

```
CUstream stream;
nvErr = NvAR_CudaStreamCreate(&stream);
nvErr = NvAR_SetCudaStream(featureHandle, NvAR_Parameter_Config(CUDAStream),
↪stream);
```

3.1.4. Summary of AR SDK Accessor Functions

Table 1-1: Summary of AR SDK Accessor Functions

Property Type	Data Type	Set and Get Accessor Function
32-bit unsigned integer	unsigned int	• NvAR_SetU32() • NvAR_GetU32()
32-bit signed integer	int	• NvAR_SetS32() • NvAR_GetS32()
Single-precision (32-bit) floating-point number	float	• NvAR_SetF32() • NvAR_GetF32()
Double-precision (64-bit) floating point number	double	• NvAR_SetF64() • NvAR_GetF64()
64-bit unsigned integer	unsigned long long	• NvAR_SetU64() • NvAR_GetU64()
Floating-point array	float*	• NvAR_SetFloatArray() • NvAR_GetFloatArray()
32-bit unsigned integer array	unsigned int*	• NvAR_SetU32Array() • NvAR_GetU32Array()
Object	void*	• NvAR_SetObject() • NvAR_GetObject()
Character string	const char*	• NvAR_SetString() • NvAR_GetString()
CUDA stream	CUstream	• NvAR_SetCudaStream() • NvAR_GetCudaStream()

3.1.5. Key Values in the Properties of a Feature Type

The key values in the properties of a feature type identify the properties that can be used with each feature type. Each key has a string equivalent and is defined by a macro that indicates the category of the property and takes a name as an input to the macro.

The following macros indicate the category of a property:

- `NvAR_Parameter_Config` indicates a configuration property.

  For more information, refer to *Configuration Properties*.

- `NvAR_Parameter_Input` indicates an input property.

  For more information, refer to *Input Properties*.

- `NvAR_Parameter_Output` indicates an output property.

  For more information, refer to *Output Properties*.

The keywords appear in various macros, depending on whether a property is an input, an output, or a configuration property.

The property type denotes the accessor functions to set and get the property, as listed in the *Summary of AR SDK Accessor Functions* table.

3.1.5.1 Configuration Properties

Here are the configuration properties in the AR SDK:

NvAR_Parameter_Config(FeatureDescription)
A description of the feature type.

String equivalent: `NvAR_Parameter_Config_FeatureDescription`
Property type: character string (`const char*`)

NvAR_Parameter_Config(CUDAStream)
The CUDA stream in which to run the feature.

String equivalent: `NvAR_Parameter_Config_CUDAStream`
Property type: CUDA stream (`CUstream`)

NvAR_Parameter_Config(ModelDir)
The path to the directory that contains both the TensorRT model files to be used to run inference for face detection or landmark detection and the `.nvf` file that contains the 3D Face model, *excluding* the model file name. For details about the format of the `.nvf` file, refer to *Appendix A: NVIDIA 3DMM File Format*.

String equivalent: `NvAR_Parameter_Config_ModelDir`
Property type: character string (`const char*`)

NvAR_Parameter_Config(ModelCacheDir)
(Windows on Arm only, such as RTX Spark) The path to the TensorRT-RTX runtime cache directory. Set this optional property before calling `NvAR_Load()`. If it is empty or is not set, the SDK uses a `cache` subdirectory under `ModelDir`.

String equivalent: `NvAR_Parameter_Config_ModelCacheDir`
Property type: character string (`const char*`)

NvAR_Parameter_Config(ModelCacheMode)
(Windows on Arm only, such as RTX Spark) Controls the TensorRT-RTX runtime cache. Set this optional property before calling `NvAR_Load()`:

- `0`: Auto (default. Load a valid cache if present; otherwise run JIT and save a new cache.
- 1: Disabled. Do not read or write cache files.
- 2: Force regenerate. Ignore any existing cache, run JIT, and write a fresh cache.

On the first load of each model without a valid cache, TensorRT-RTX can require on the order of several seconds for JIT compilation. Later loads reuse the cached kernels when `ModelCacheMode` is Auto.

String equivalent: `NvAR_Parameter_Config_ModelCacheMode`
Property type: unsigned integer

NvAR_Parameter_Config(BatchSize)
The number of inferences to be run at one time on the GPU.

String equivalent: `NvAR_Parameter_Config_BatchSize`
Property type: unsigned integer

NvAR_Parameter_Config(Landmarks_Size)
The length of the output buffer that contains the x and y coordinates in pixels of the detected landmarks. This property applies only to the landmark detection feature.

String equivalent: `NvAR_Parameter_Config_Landmarks_Size`
Property type: unsigned integer

NvAR_Parameter_Config(LandmarksConfidence_Size)
The length of the output buffer that contains the confidence values of the detected landmarks. This property applies only to the landmark detection feature.

String equivalent: `NvAR_Parameter_Config_LandmarksConfidence_Size`
Property type: unsigned integer

NvAR_Parameter_Config(Temporal)
Flag to enable optimization for temporal input frames. Enable the flag when the input is a video.

String equivalent: `NvAR_Parameter_Config_Temporal`
Property type: unsigned integer

NvAR_Parameter_Config(ShapeEigenValueCount)
The number of eigenvalues used to describe shape. In the supplied `face_model2.nvf` are 100

shapes (also known as identity) eigenvalues, but `ShapeEigenValueCount` should be queried when you allocate an array to receive the eigenvalues.

String equivalent: `NvAR_Parameter_Config_ShapeEigenValueCount`
Property type: unsigned integer

NvAR_Parameter_Config(ExpressionCount)
The number of coefficients used to represent expression. In the supplied `face_model2.nvf` are 53 expression blendshape coefficients, but `ExpressionCount` should be queried when allocating an array to receive the coefficients.

String equivalent: `NvAR_Parameter_Config_ExpressionCount`
Property type: unsigned integer

NvAR_Parameter_Config(UseCudaGraph)
Flag to enable CUDA Graph optimization. CUDA Graphs reduce the overhead of GPU operation submission for features such as Eye Contact (Gaze Redirection) and 3D Body Pose tracking. Enabling this flag is recommended on Windows on Arm (for example, RTX Spark) for better steady-state performance. The default is false.

String equivalent: `NvAR_Parameter_Config_UseCudaGraph`
Property type: bool

NvAR_Parameter_Config(GazeRedirect)
Flag to enable the redirection of gaze, in addition to gaze estimation, in the eye contact feature. By default, this is set to true.

String equivalent: `NvAR_Parameter_Config_GazeRedirect`
Property type: bool

NvAR_Parameter_Config(Mode)
Mode to select High Performance or High Quality for 3D Body Pose or Facial Landmark Detection.

String equivalent: `NvAR_Parameter_Config_Mode`
Property type: unsigned integer

NvAR_Parameter_Config(MaxBoxesPerFrame)
Maximum number of tracked body boxes that are accepted per frame. For 3D Body Pose Estimation, this is a read-only capability value.

String equivalent: `NvAR_Parameter_Config_MaxBoxesPerFrame`
Property type: unsigned integer

NvAR_Parameter_Config(EnableContact)
Enables or disables contact correction and contact-driven inverse kinematics (IK) for 3D Body Pose Estimation. Set this property before `NvAR_Load`. The default is 1. Keep contact correction enabled for fixed-camera scenes and whenever contact stability matters. Disable it when the camera moves, or when lower compute cost matters more than contact stabilization, because disabling it can increase visible drift and sliding.

String equivalent: `NvAR_Parameter_Config_EnableContact`
Property type: unsigned integer

NvAR_Parameter_Config(ReferencePose)
CPU buffer of type *NvAR_Point3f* to hold the Reference Pose for Joint Rotations for 3D Body Pose.

String equivalent: `NvAR_Parameter_Config_ReferencePose`
Property type: object (`void*`)

NvAR_Parameter_Config(FullBodyOnly)
Flag to select pose estimation mode in 3D body pose and body detection: Full Body only or Full and upper body pose estimation mode.

- 0: set to Full and upper body pose estimation. Supports only high quality mode in 3D body pose.
- 1: set to Full body pose estimation. Supports both high quality and high performance modes in 3D body pose.

String equivalent: `NvAR_Parameter_Config_FullBodyOnly`
Property type: unsigned int

NvAR_Parameter_Config(PostprocessJointAngle)
Flag to enable or disable the postprocessing steps for joint angles corresponding to the joints predicted with low confidence in 3D body pose. To be used only when `FullBodyOnly` is set to 0. We recommend that you set this to true when input is upper body image or video.

String equivalent: `NvAR_Parameter_Config_PostprocessJointAngle`
Property type: bool

NvAR_Parameter_Config(TargetSeatedPoseForInterpolation)
CPU buffer of type *NvAR_Quaternion* to hold the target seated pose to be used for post-processing joint rotations for 3D Body Pose. For the joints that are predicted with low confidence, the output pose is interpolated to the corresponding pose specified in this target pose. To be used only when `FullBodyOnly` is set to 0.

String equivalent: `NvAR_Parameter_Config_TargetSeatedPoseForInterpolation`

Property type: object (`void*`)

NvAR_Parameter_Config(TargetStandPoseForInterpolation)
CPU buffer of type *NvAR_Quaternion* to hold the target standing pose to be used for post-processing joint rotations for 3D Body Pose. For the joints that are predicted with low confidence, the output pose is interpolated to the corresponding pose specified in this target pose. To be used only when `FullBodyOnly` is set to 0.

String equivalent: `NvAR_Parameter_Config_TargetStandPoseForInterpolation`
Property type: object (`void*`)

NvAR_Parameter_Config(TrackPeople)
Flag to select Multi-Person Tracking for 3D Body Pose Tracking.

String equivalent: `NvAR_Parameter_Config_TrackPeople`
Property type: unsigned integer

NvAR_Parameter_Config(ShadowTrackingAge)
The age after which the multi-person tracker no longer tracks the object in shadow mode. This property is measured in the number of frames.

String equivalent: `NvAR_Parameter_Config_ShadowTrackingAge`
Property type: unsigned integer

NvAR_Parameter_Config(ProbationAge)
The age after which the multi-person tracker marks the object valid and assigns an ID for tracking. This property is measured in the number of frames.

String equivalent: `NvAR_Parameter_Config_ProbationAge`
Property type: unsigned integer

NvAR_Parameter_Config(NetworkOutputImgWidth)
Width of the output image generated from the network (512 or 1024).

String equivalent: `NvAR_Parameter_Config_NetworkOutputImgWidth`
Property type: unsigned integer

NvAR_Parameter_Config(NetworkOutputImgHeight)
Height of the output image generated from the network (512 or 1024).

String equivalent: `NvAR_Parameter_Config_NetworkOutputImgHeight`

Property type: unsigned integer

NvAR_Parameter_Config(MaxTargetsTracked)
The maximum number of targets to be tracked by the multi-person tracker. After this limit is met, any new targets are discarded.

String equivalent: `NvAR_Parameter_Config_MaxTargetsTracked`
Property type: unsigned integer

3.1.5.2 Input Properties

Here are the input properties in the AR SDK:

NvAR_Parameter_Input(Image)
GPU input image buffer of type `NvCVImage`.

String equivalent: `NvAR_Parameter_Input_Image`
Property type: object (`void*`)

NvAR_Parameter_Input(Width)
The width of the input image buffer in pixels.

String equivalent: `NvAR_Parameter_Input_Width`
Property type: integer

NvAR_Parameter_Input(Height)
The height of the input image buffer in pixels.

String equivalent: `NvAR_Parameter_Input_Height`
Property type: integer

NvAR_Parameter_Input(Landmarks)
CPU input array of type *NvAR_Point2f* that contains the facial landmark points.

String equivalent: `NvAR_Parameter_Input_Landmarks`
Property type: object (`void*`)

NvAR_Parameter_Input(BoundingBoxes)
Bounding boxes that determine the region of interest (ROI) of an input image that contains a face of type *NvAR_BBoxes*.

String equivalent: `NvAR_Parameter_Input_BoundingBoxes`

Property type: object (`void*`)

NvAR_Parameter_Input(FocalLength)
The focal length of the camera used for 3D Body Pose.

String equivalent: `NvAR_Parameter_Input_FocalLength`
Property type: float

NvAR_Parameter_Input(TrackingBoundingBoxes)
Caller-provided tracking bounding boxes for 3D Body Pose Estimation.

String equivalent: `NvAR_Parameter_Input_TrackingBoundingBoxes`
Property type: object (`void*`)

NvAR_Parameter_Input(Flush)
Flush mode for the current 3D Body Pose Estimation stream. At the end of the stream, set this property to 1 and keep calling `NvAR_Run` until `NvAR_Parameter_Output(StreamFlushed)` returns 1.

String equivalent: `NvAR_Parameter_Input_Flush`
Property type: unsigned integer

3.1.5.3 Output Properties

Here are the output properties in the AR SDK:

NvAR_Parameter_Output(BoundingBoxes)
CPU output bounding boxes of type *NvAR_BBoxes*.

String equivalent: `NvAR_Parameter_Output_BoundingBoxes`
Property type: object (`void*`)

NvAR_Parameter_Output(TrackingBoundingBoxes)
CPU output tracking bounding boxes of type *NvAR_TrackingBBoxes*.

String equivalent: `NvAR_Parameter_Output_TrackingBBoxes`
Property type: object (`void*`)

NvAR_Parameter_Output(Ready)
Read-only output status for 3D Body Pose Estimation. A value of 1 means the registered output buffers contain a valid output frame.

String equivalent: `NvAR_Parameter_Output_Ready`
Property type: unsigned integer

NvAR_Parameter_Output(StreamFlushed)
Read-only flush completion status for the current 3D Body Pose Estimation stream.

String equivalent: `NvAR_Parameter_Output_StreamFlushed`
Property type: unsigned integer

NvAR_Parameter_Output(NumBodies)
Read-only number of valid bodies in the 3D Body Pose Estimation output frame that is ready.

String equivalent: `NvAR_Parameter_Output_NumBodies`
Property type: unsigned integer

NvAR_Parameter_Output(BoundingBoxesConfidence)
Float array of confidence values for each returned bounding box.

String equivalent: `NvAR_Parameter_Output_BoundingBoxesConfidence`
Property type: floating point array

NvAR_Parameter_Output(Landmarks)
CPU output buffer of type *NvAR_Point2f* to hold the output detected landmark key points. For more information, refer to Facial point annotations. The order of the points in the CPU buffer follows the order in MultiPIE 68-point markups, and the 126 points cover more points along the cheeks, the eyes, and the laugh lines.

String equivalent: `NvAR_Parameter_Output_Landmarks`
Property type: object (`void*`)

NvAR_Parameter_Output(LandmarksConfidence)
Float array of confidence values for each detected landmark point.

String equivalent: `NvAR_Parameter_Output_LandmarksConfidence`
Property type: floating point array

NvAR_Parameter_Output(Pose)
CPU array of type *NvAR_Quaternion* to hold the output-detected pose as an XYZW quaternion.

String equivalent: `NvAR_Parameter_Output_Pose`
Property type: object (`void*`)

NvAR_Parameter_Output(FaceMesh)
CPU 3D face Mesh of type `NvAR_FaceMesh`.

String equivalent: `NvAR_Parameter_Output_FaceMesh`
Property type: object (`void*`)

NvAR_Parameter_Output(RenderingParams)
CPU output structure of type *NvAR_RenderingParams* that contains the rendering parameters that might be used to render the 3D face mesh.

String equivalent: `NvAR_Parameter_Output_RenderingParams`
Property type: object (`void*`)

NvAR_Parameter_Output(ShapeEigenValues)
Float array of shape eigenvalues. Get `NvAR_Parameter_Config(ShapeEigenValueCount)` to determine the count of eigenvalues.

String equivalent: `NvAR_Parameter_Output_ShapeEigenValues`
Property type: floating point array

NvAR_Parameter_Output(ExpressionCoefficients)
Float array of expression coefficients. Get `NvAR_Parameter_Config(ExpressionCount)` to determine the count of coefficients.

String equivalent: `NvAR_Parameter_Output_ExpressionCoefficients`
Property type: floating point array

NvAR_Parameter_Output(KeyPoints)
CPU output buffer of type *NvAR_Point2f* to hold the output detected 2D Keypoints for Body Pose. For more information about the Keypoint names and the order of Keypoint output, refer to *Appendix B: 3D Body Pose Keypoint Format*.

String equivalent: `NvAR_Parameter_Output_KeyPoints`
Property type: object (`void*`)

NvAR_Parameter_Output(KeyPoints3D)
CPU output buffer of type *NvAR_Point3f* to hold the output detected 3D Keypoints for Body Pose. For more information about the Keypoint names and the order of Keypoint output, refer to *Appendix B: 3D Body Pose Keypoint Format*.

String equivalent: `NvAR_Parameter_Output_KeyPoints3D`
Property type: object (`void*`)

NvAR_Parameter_Output(RestPose)
CPU output buffer of type *NvAR_Point3f* to hold 3D Body Pose Estimation rest-pose data in meters using the SDK cumulative direct-rest representation. For the T-pose derivation and an example that reconstructs the emitted keypoints, refer to *Reconstructing T-Pose and Posed 3D Keypoints*.

String equivalent: `NvAR_Parameter_Output_RestPose`
Property type: object (`void*`)

NvAR_Parameter_Output(RootPose)
CPU output buffer of type *NvAR_Transform3f* to hold the root pose for each tracked body. The translation is the root keypoint in camera coordinates, in meters. The rotation rotates root-relative direct-rest points into the camera coordinate system. For T-pose retargeting, derive the T-pose-basis root rotation as shown in *Reconstructing T-Pose and Posed 3D Keypoints*.

String equivalent: `NvAR_Parameter_Output_RootPose`
Property type: object (`void*`)

NvAR_Parameter_Output(JointRotations)
CPU output buffer of type *NvAR_Quaternion* to hold local 3D Body Pose Estimation joint rotations as XYZW quaternions in SDK skeleton order and in the SDK direct-rest basis. For T-pose and posed 3D keypoint reconstruction formulas, refer to *Reconstructing T-Pose and Posed 3D Keypoints*.

String equivalent: `NvAR_Parameter_Output_JointRotations`
Property type: object (`void*`)

NvAR_Parameter_Output(JointAngles)
CPU output buffer of type *NvAR_Quaternion* to hold the local rotation of each joint as an XYZW quaternion. This parameter is a compatibility alias for `NvAR_Parameter_Output(JointRotations)`.

String equivalent: `NvAR_Parameter_Output_JointAngles`
Property type: object (`void*`)

NvAR_Parameter_Output(KeyPointsConfidence)
Float array of confidence values for each detected keypoint.

String equivalent: `NvAR_Parameter_Output_KeyPointsConfidence`
Property type: floating point array

NvAR_Parameter_Output(OutputHeadTranslation)
Float array of three values that represent the x, y and z values of head translation with respect

to the camera for Eye Contact.

String equivalent: `NvAR_Parameter_Output_OutputHeadTranslation`
Property type: floating point array

NvAR_Parameter_Output(OutputGazeVector)
Float array of two values that represent the yaw and pitch angles of the estimated gaze for Eye Contact.

String equivalent: `NvAR_Parameter_Output_OutputGazeVector`
Property type: floating point array

NvAR_Parameter_Output(HeadPose)
CPU array of type *NvAR_Quaternion* to hold the output-detected head pose as an XYZW quaternion in Eye Contact. This is an alternative to the head pose that was obtained from the facial landmarks feature. This head pose is obtained using the PnP algorithm over the landmarks.

String equivalent: `NvAR_Parameter_Output_HeadPose`
Property type: object (`void*`)

NvAR_Parameter_Output(GazeDirection)
Float array of two values that represent the yaw and pitch angles of the estimated gaze for Eye Contact.

String equivalent: `NvAR_Parameter_Output_GazeDirection`
Property type: floating point array

3.1.6. Getting the Value of a Property of a Feature

To get the value of a property of a feature, call the get accessor function that is appropriate for the data type of the property. In the call to the function, pass the following information:

- The feature handle to the feature instance.
- The key value that identifies the property that you are getting.
- The location in memory where you want the value of the property to be written.

The following example determines the length of the `NvAR_Point2f` output buffer that was returned by the landmark detection feature:

```
unsigned int OUTPUT_SIZE_KPTS;

NvAR_GetU32(landmarkDetectHandle, NvAR_Parameter_Config(Landmarks_Size),
&OUTPUT_SIZE_KPTS);
```

3.1.7. Setting a Property for a Feature

To set a property for a feature:

1. Allocate memory for all inputs and outputs that are required by the feature and any other properties that might be required.
2. Call the set accessor function that is appropriate for the data type of the property.

   In the call to the function, pass the following information:

   - The feature handle to the feature instance.
   - The key value that identifies the property that you are setting.
   - A pointer to the value to which you want to set the property.

   The following example sets the file path to the file that contains the output 3D face model:

```
const char *modelPath = "file/path/to/model";

NvAR_SetString(landmarkDetectHandle, NvAR_Parameter_Config(ModelDir),
modelPath);
```

   The following example sets up the input image buffer in GPU memory, which is required by the face detection feature:

   Note

   The example sets up an 8-bit chunky/interleaved BGR array.

```
NvCVImage InputImageBuffer;

NvCVImage_Alloc(&inputImageBuffer, input_image_width,
input_image_height, NVCV_BGR, NVCV_U8, NVCV_CHUNKY, NVCV_GPU, 1) ;

NvAR_SetObject(landmarkDetectHandle, NvAR_Parameter_Input(Image),
&InputImageBuffer, sizeof(NvCVImage));
```

For more information about the properties and the input and output requirements for each feature, refer to *Properties for the AR SDK Features*.

Note

The listed property name is the input to the macro that defines the key value for the property.

3.1.8. Loading a Feature Instance

You can load the feature after setting the configuration properties that are required to load an instance of a feature type.

To load a feature instance, call the *NvAR_Load* function and specify the handle that was created for the feature instance when the instance was created. For more information, refer to *Creating an Instance of a Feature Type*.

The following example loads an instance of the face detection feature type:

```
NvAR_Load(faceDetectHandle);
```

3.1.9. Running a Feature Instance

Before you can run the feature instance, you must load an instance of a feature type and set the user-allocated input and output memory buffers that are required when the feature instance is run.

To run a feature instance, call the *NvAR_Run* function and specify the handle that was created for the feature instance when the instance was created. For more information, refer to *Creating an Instance of a Feature Type*.

The following example shows how to run a face detection feature instance:

```
NvAR_Run(faceDetectHandle);
```

3.1.10. Resetting a Feature Instance

When a feature instance needs to be reset to its original state, as if it is the first frame which is being run on the instance, the reset functionality can be used. This does not reset the stream and config values but only resets those variables that have a temporal dependency with the previous frames. Currently, the reset functionality applies only to the Eye Contact feature and throws an exception with the other features. Note that the state handle parameter needs to be a nullptr for the feature to be reset.

The following example shows how to reset the Eye Contact feature instance:

```
NvAR_ResetState(eyeContactHandle, nullptr);
```

3.1.11. Destroying a Feature Instance

When a feature instance is no longer required, you need to destroy it to free the resources and memory that the feature instance allocated internally. Memory buffers are provided as input and to hold the output of a feature and must be separately deallocated.

To destroy a feature instance, call the *NvAR_Destroy* function and specify the handle that was created for the feature instance when the instance was created. For more information, refer to *Creating an Instance of a Feature Type*.

```
NvAR_Destroy(faceDetectHandle);
```

3.2. Working with Image Frames on GPU or CPU Buffers

Effect filters accept image buffers as `NvCVImage` objects. The image buffers can be CPU or GPU buffers, but for performance reasons, the effect filters require GPU buffers. The AR SDK provides functions for converting an image representation to an `NvCVImage` object and transferring images between CPU and GPU buffers.

For more information about NvCVImage, refer to the NvCVImage API Guide. This section provides a synopsis of the functions that are most frequently used with the AR SDK.

3.2.1. Converting Image Representations to NvCVImage Objects

The SDK provides functions for converting OpenCV images and other image representations to NvCVImage objects. Each function places a wrapper around an existing buffer. The wrapper prevents the buffer from being freed when the destructor of the wrapper is called.

3.2.1.1 Converting OpenCV Images to NvCVImage Objects

You can use the wrapper functions that the SDK provides specifically for RGB OpenCV images.

Note

The AR SDK provides wrapper functions only for RGB images. No wrapper functions are provided for YUV images.

- To create an NvCVImage object wrapper for an OpenCV image, use the NVWrapperForCVMat() function.

```
//Allocate source and destination OpenCV images
cv::Mat srcCVImg( );
cv::Mat dstCVImg(...);

// Declare source and destination NvCVImage objects
NvCVImage srcCPUImg;
NvCVImage dstCPUImg;

NVWrapperForCVMat(&srcCVImg, &srcCPUImg);
NVWrapperForCVMat(&dstCVImg, &dstCPUImg);
```

- To create an OpenCV image wrapper for an NvCVImage object, use the CVWrapperForNvCVImage() function.

```
// Allocate source and destination NvCVImage objects
NvCVImage srcCPUImg(...);
NvCVImage dstCPUImg(...);

//Declare source and destination OpenCV images
cv::Mat srcCVImg;
cv::Mat dstCVImg;

CVWrapperForNvCVImage(&srcCPUImg, &srcCVImg);
CVWrapperForNvCVImage(&dstCPUImg, &dstCVImg);
```

3.2.1.2 Converting Other Image Representations to NvCVImage Objects

To convert other image representations, call the NvCVImage_Init() function to place a wrapper around an existing buffer (`srcPixelBuffer`).

```
NvCVImage src_gpu;

vfxErr = NvCVImage_Init(&src_gpu, 640, 480, 1920, srcPixelBuffer,
NVCV_BGR, NVCV_U8, NVCV_INTERLEAVED, NVCV_GPU);

NvCVImage src_cpu;

vfxErr = NvCVImage_Init(&src_cpu, 640, 480, 1920, srcPixelBuffer,
NVCV_BGR, NVCV_U8, NVCV_INTERLEAVED, NVCV_CPU);
```

3.2.1.3 Converting Decoded Frames from the NvDecoder to NvCVImage Objects

To convert decoded frames from the NVDecoder to `NvCVImage` objects, call the NvCVImage_Transfer() function to convert the decoded frame that is provided by the NvDecoder from the decoded pixel format to the format that is required by a feature of the AR SDK.

The following sample shows a decoded frame that was converted from the NV12 frame format to the BGRA pixel format. NV12 frame format to the BGRA pixel format.

```
NvCVImage decoded_frame, BGRA_frame, stagingBuffer;

NvDecoder dec;

//Initialize decoder

//Assuming dec.GetOutputFormat() == cudaVideoSurfaceFormat_NV12

//Initialize memory for decoded frame
NvCVImage_Init(&decoded_frame, dec.GetWidth(), dec.GetHeight(),
               dec.GetDeviceFramePitch(), NULL, NVCV_YUV420, NVCV_U8, NVCV_
↪NV12,
               NVCV_GPU, 1);

decoded_frame.colorSpace = NVCV_709 | NVCV_VIDEO_RANGE |
                           NVCV_CHROMA_COSITED;

//Allocate memory for BGRA frame, and set alpha opaque
NvCVImage_Alloc(&BGRA_frame, dec.GetWidth(), dec.GetHeight(), NVCV_BGRA,
                NVCV_U8, NVCV_CHUNKY, NVCV_GPU, 1);

cudaMemset(BGRA_frame.pixels, -1, BGRA_frame.pitch *
           BGRA_frame.height);

decoded_frame.pixels = (void*)dec.GetFrame();

//Convert from decoded frame format(NV12) to desired format(BGRA)
NvCVImage_Transfer(&decoded_frame, &BGRA_frame, 1.f, stream, &stagingBuffer);
```

Note

The preceding sample assumes the typical colorspace specification for HD content. SD typically uses NVCV_601. There are eight possible combinations, and you should use the one that matches your video as described in the video header or proceed by trial and error.

Here is some additional information:

- If the colors are incorrect, swap 709<->601.
- If they are washed out or blown out, swap VIDEO<->FULL.
- If the colors are shifted horizontally, swap INTSTITIAL<->COSITED.

3.2.1.4 Converting an NvCVImage Object to a Buffer Encodable by NvEncoder

To convert the NvCVImage object to the pixel format that is used during encoding via NvEncoder, if necessary, call the NvCVImage_Transfer() function.

The following sample shows a frame that is encoded in the BGRA pixel format.

```
//BGRA frame is 4-channel, u8 buffer residing on the GPU
NvCVImage BGRA_frame;
NvCVImage_Alloc(&BGRA_frame, dec.GetWidth(), dec.GetHeight(),
                NVCV_BGRA, NVCV_U8, NVCV_CHUNKY, NVCV_GPU, 1);

//Initialize encoder with a BGRA output pixel format
using NvEncCudaPtr = std::unique_ptr<NvEncoderCuda,
                     std::function<void(NvEncoderCuda*)>>;

NvEncCudaPtr pEnc(new NvEncoderCuda(cuContext, dec.GetWidth(),
                  dec.GetHeight(), NV_ENC_BUFFER_FORMAT_ARGB));

pEnc->CreateEncoder(&initializeParams);

// your code here

std::vector<std::vector<uint8_t>> vPacket;

//Get the address of the next input frame from the encoder
const NvEncInputFrame* encoderInputFrame = pEnc->GetNextInputFrame();

//Copy the pixel data from BGRA_frame into the input frame address obtained
 ↪above
NvEncoderCuda::CopyToDeviceFrame(cuContext,
                                 BGRA_frame.pixels,
                                 BGRA_frame.pitch,
                                 (CUdeviceptr)encoderInputFrame->inputPtr,
                                 encoderInputFrame->pitch,
                                 pEnc->GetEncodeWidth(),
                                 pEnc->GetEncodeHeight(),
                                 CU_MEMORYTYPE_DEVICE,
                                 encoderInputFrame->bufferFormat,
                                 encoderInputFrame->chromaOffsets,
                                 encoderInputFrame->numChromaPlanes);
```

(continues on next page)

(continued from previous page)

```
pEnc->EncodeFrame(vPacket);
```

3.2.2. Allocating an NvCVImage Object Buffer

You can allocate the buffer for an `NvCVImage` object by using the `NvCVImage` allocation constructor or image functions. In both options, the buffer is automatically freed by the destructor when the images go out of scope.

3.2.2.1 Using the NvCVImage Allocation Constructor to Allocate a Buffer

The `NvCVImage` allocation constructor creates an object to which memory has been allocated and that has been initialized. For more information, refer to NvCVImage Allocation Constructor.

The final three optional parameters of the allocation constructor determine the properties of the resulting `NvCVImage` object:

- The pixel organization determines whether blue, green, and red are in separate planes or interleaved.
- The memory type determines whether the buffer resides on the GPU or the CPU.
- The byte alignment determines the gap between consecutive scanlines.

The following examples show how to use the final three optional parameters of the allocation constructor to determine the properties of the `NvCVImage` object.

This example creates an object without setting the final three optional parameters of the allocation constructor. In this object, the blue, green, and red components interleaved in each pixel, the buffer resides on the CPU, and the byte alignment is the default alignment.

```
NvCVImage cpuSrc(
    srcWidth,
    srcHeight,
    NVCV_BGR,
    NVCV_U8
);
```

This example creates an object with identical pixel organization, memory type, and byte alignment to the previous example by setting the final three optional parameters explicitly. As in the previous example, the blue, green, and red components are interleaved in each pixel, the buffer resides on the CPU, and the byte alignment is the default—that is, optimized for maximum performance.

```
NvCVImage src(
    srcWidth,
    srcHeight,
    NVCV_BGR,
    NVCV_U8,
    NVCV_INTERLEAVED,
    NVCV_CPU,
    0
);
```

This example creates an object in which the blue, green, and red components are in separate planes, the buffer resides on the GPU, and the byte alignment ensures that no gap exists between one scanline and the next scanline.

```
NvCVImage gpuSrc(
    srcWidth,
    srcHeight,
    NVCV_BGR,
    NVCV_U8,
    NVCV_PLANAR,
    NVCV_GPU,
    1
);
```

3.2.2.2 Using Image Functions to Allocate a Buffer

By declaring an empty image, you can defer buffer allocation.

1. Declare an empty `NvCVImage` object.

   ```
   NvCVImage xfr;
   ```

2. Allocate or reallocate the buffer for the image.

- To allocate the buffer, call the NvCVImage_Alloc() function.

  Allocate a buffer this way when the image is part of a state structure, where you do not know the size of the image until later.

- To reallocate a buffer, call NvCVImage_Realloc().

  This function checks for an allocated buffer and reshapes the buffer if it is big enough before freeing the buffer and calling NvCVImage_Alloc().

3.2.3. Transferring Images Between CPU and GPU Buffers

If the memory types of the input and output image buffers are different, an application can transfer images between CPU and GPU buffers.

3.2.3.1 Transferring Input Images from a CPU Buffer to a GPU Buffer

1. Create an `NvCVImage` object to use as a staging GPU buffer that has the same dimensions and format as the source CPU buffer.

   ```
   NvCVImage srcGpuPlanar(inWidth, inHeight, NVCV_BGR, NVCV_F32,
                          NVCV_PLANAR, NVCV_GPU, 1);
   ```

2. Create a staging buffer in one of the following ways:

   - To avoid allocating memory in a video pipeline, create a GPU buffer that has the same dimensions and format as required for input to the video effect filter.

     ```
     NvCVImage srcGpuStaging(inWidth, inHeight, srcCPUImg.pixelFormat,
                             srcCPUImg.componentType, srcCPUImg.planar,
     ↪NVCV_GPU);
     ```

- To simplify your application program code, declare an empty staging buffer.

```
NvCVImage srcGpuStaging;
```

  An appropriate buffer will be allocated or reallocated as needed.

3. Call the NvCVImage_Transfer() function to copy the source CPU buffer contents into the final GPU buffer via the staging GPU buffer.

```
//Read the image into srcCPUImg

NvCVImage_Transfer(&srcCPUImg, &srcGPUPlanar, 1.0f, stream,
                   &srcGPUStaging);
```

3.2.3.2 Transferring Output Images from a GPU Buffer to a CPU Buffer

1. Create an NvCVImage object to use as a staging GPU buffer that has the same dimensions and format as the destination CPU buffer.

```
NvCVImage dstGpuPlanar(outWidth, outHeight, NVCV_BGR, NVCV_F32,
                       NVCV_PLANAR, NVCV_GPU, 1);
```

2. Create a staging buffer in one of the following ways:

   - To avoid allocating memory in a video pipeline, create a GPU buffer that has the same dimensions and format as the output of the video effect filter.

```
NvCVImage dstGpuStaging(outWidth, outHeight, dstCPUImg.pixelFormat,
                        dstCPUImg.componentType, dstCPUImg.planar,
↪NVCV_GPU);
```

   - To simplify your application program code, declare an empty staging buffer.

```
NvCVImage dstGpuStaging;
```

     An appropriately sized buffer will be allocated as needed.

3. Call the NvCVImage_Transfer() function to copy the GPU buffer contents into the destination CPU buffer via the staging GPU buffer.

```
//Retrieve the image from the GPU to CPU, perhaps with conversion.
NvCVImage_Transfer(&dstGpuPlanar, &dstCPUImg, 1.0f, stream,
                   &dstGpuStaging);
```

3.3. Properties for the AR SDK Features

This section provides the properties and their values for the features in the AR SDK.

3.3.1. Face Tracking Property Values

The following tables list the values for the configuration, input, and output properties for face tracking.

Table 3-1: Configuration Properties for Face Tracking

Property Name	Value
Feature-Description	String is free-form text that describes the feature. The string is set by the SDK and cannot be modified by the user.
CUDAStream	The CUDA stream, which is set by the user.
ModelDir	String that contains the path to the folder that contains the TensorRT package files. Set by the user.
ModelCacheDir	**(Windows on Arm only, such as RTX Spark)** Optional string path to the model runtime cache directory. Empty or unset selects `ModelDir/cache`. Set by the user before `NvAR_Load()`.
Model-CacheMode	**(Windows on Arm only, such as RTX Spark)** Optional unsigned integer that controls the model runtime cache: `0` = Auto (default), 1 = Disabled, 2 = ForceRegenerate. Set by the user before `NvAR_Load()`.
Temporal	Unsigned integer to enable (1) or disable (0) the temporal optimization of face detection. If enabled, only one face is returned. For more information, refer to *Face Detection and Tracking*. Set by the user.

Table 3-2: Input Properties for Face Tracking

Property Name	Value
Image	Interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. To be allocated and set by the user.

Table 3-3: Output Properties for Face Tracking

Property Name	Value
BoundingBoxes	`NvAR_BBoxes` structure that holds the detected face boxes. To be allocated by the user.
BoundingBoxesConfidence	Optional: An array of single-precision (32-bit) floating-point numbers that contain the confidence values for each detected face box. To be allocated by the user.

3.3.2. Landmark Tracking Property Values

The following tables list the values for the configuration, input, and output properties for landmark tracking.

Table 3-4: Configuration Properties for Landmark Tracking

Property Name	Value
FeatureDescription	String that describes the feature.
CUDAStream	The CUDA stream. Set by the user.
ModelDir	String that contains the path to the folder that contains the TensorRT package files. Set by the user.
ModelCacheDir	**(Windows on Arm only, such as RTX Spark)** Optional string path to the model runtime cache directory. Empty or unset selects `ModelDir/cache`. Set by the user before `NvAR_Load()`.
ModelCacheMode	**(Windows on Arm only, such as RTX Spark)** Optional unsigned integer that controls the model runtime cache: `0` = Auto (default), 1 = Disabled, 2 = ForceRegenerate. Set by the user before `NvAR_Load()`.
BatchSize	The number of inferences to be run at one time on the GPU. The maximum value is 8. Temporal optimization of landmark detection is supported only for `BatchSize=1`.
Landmarks_Size	Unsigned integer, 68 or 126. Specifies the number of landmark points (x and y values) to be returned. Set by the user.
LandmarksConfidence_Size	Unsigned integer, 68 or 126. Specifies the number of landmark confidence values for the detected keypoints to be returned. Set by the user.
Temporal	Unsigned integer to enable (1) or disable (0) the temporal optimization of landmark detection. If enabled, only one input bounding box is supported as the input. For more information, refer to *Face Detection and Tracking*. Set by the user.
Mode	Optional: Unsigned integer. Set 0 to enable Performance mode (default) or 1 to enable Quality mode for landmark detection. Set by the user.

Table 3-5: Input Properties for Landmark Tracking

Property Name	Value
Image	Interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. To be allocated and set by the user.
Bounding-Boxes	Optional: `NvAR_BBoxes` structure that contains the number of bounding boxes that are equal to `BatchSize` on which to run landmark detection. If not specified as an input property, face detection is automatically run on the input image. For more information, refer to *Face Detection and Tracking*. To be allocated by the user.

Table 3-6: Output Properties for Landmark Tracking

Property Name	Value
Landmarks	`NvAR_Point2f` array, which must be large enough to hold the number of points given by the product of `NvAR_Parameter_Config(BatchSize)` and `NvAR_Parameter_Config(Landmarks_Size)`. To be allocated by the user.
Pose	Optional: `NvAR_Quaternion` array, which must be large enough to hold the number of quaternions equal to `NvAR_Parameter_Config(BatchSize)`. The OpenGL standards coordinate convention is used: When you look up from a camera, the coordinates are x (camera right), y (camera up), and z (toward camera). To be allocated by the user.
LandmarksConfidence	Optional: An array of single-precision (32-bit) floating-point numbers, which must be large enough to hold the number of confidence values given by the product of the following: • `NvAR_Parameter_Config(BatchSize)` • `NvAR_Parameter_Config(LandmarksConfidence_Size` To be allocated by the user.
BoundingBoxes	Optional: `NvAR_BBoxes` structure that contains the detected face through face detection performed by the landmark detection feature. For more information, refer to *Face Detection and Tracking*. To be allocated by the user.

3.3.3. Eye Contact Property Values

The following tables list the values for the configuration, input, and output properties for gaze redirection.

Table 3-7: Configuration Properties for Eye Contact

Property Name	Value
FeatureDescription	String that describes the feature.
ModelDir	String that contains the path to the folder that contains the TensorRT package files. Set by the user.
ModelCacheDir	**(Windows on Arm only, such as RTX Spark)** Optional string path to the model runtime cache directory. Empty or unset selects `ModelDir/cache`. Set by the user before `NvAR_Load()`.
ModelCacheMode	**(Windows on Arm only, such as RTX Spark)** Optional unsigned integer that controls the model runtime cache: `0` = Auto (default), `1` = Disabled, `2` = ForceRegenerate. Set by the user before `NvAR_Load()`.
BatchSize	The number of inferences to be run at one time on the GPU. The maximum value is 1.
Landmarks_Size	Unsigned integer, either 68 or 126. Specifies the number of landmark points (x and y values) to be returned. Set by the user.
LandmarksConfidence_Size	Unsigned integer, either 68 or 126. Specifies the number of landmark confidence values for the detected keypoints to be returned. Set by the user.
GazeRedirect	Flag to enable or disable gaze redirection. When enabled, the gaze is estimated, and the redirected image is set as the output. When disabled, the gaze is estimated but redirection does not occur.
Temporal	Unsigned integer to enable (1) or disable (0) the temporal optimization of landmark detection. Set by the user.
DetectClosure	Flag to toggle the detection of eye closure and occlusion. The default value is **On**.
EyeSizeSensitivity	An unsigned integer in the range 2–5, inclusive, that is used to increase the sensitivity of the algorithm to the redirected eye size. A value of 2 uses a smaller eye region, and a value of 5 uses a larger eye size.
UseCudaGraph	Bool. Default is False. Flag to use CUDA Graphs for optimization. Recommended on Windows on Arm (for example, RTX Spark) for better steady-state performance. Set by the user.
EnableLookAway	Bool. Default is false. Flag that, when set to true, redirects the eyes to look away at a random time for a random period. The eyes follow the relative changes in estimated gaze during the lookaway period. Set by the user.
LookAwayOffsetMax	Unsigned int value in the range 0–10. Default is 5. If the value is set to *x* degrees, a randomly chosen offset angle in the range −*x* to *x* in degrees will be added to the lookaway angle during the random lookaway period. The lookaway angle is based on the relative motion of the eyes in the input image during the lookaway period. It is not used outside the lookaway period. Set by the user.
LookAwayIntervalMin	Unsigned int value in the range 1–600. Default is 100. Minimum limit for the number of frames at which random look away occurs. This value is applicable only when `EnableLookAway` is set to true. Set by the user.

Table 3-8: Input Properties for Eye Contact

Property Name	Value
Image	Interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. To be allocated and set by the user.
Width	The width of the input image buffer that contains the face to which the face model will be fitted. Set by the user.
Height	The height of the input image buffer that contains the face to which the face model will be fitted. Set by the user.
Landmarks	Optional: An `NvAR_Point2f` array that contains the landmark points of size `NvAR_Parameter_Config(Landmarks_Size)` that is returned by the landmark detection feature. If landmarks are not provided to this feature, an input image must be provided. To be allocated by the user.

Table 3-9: Output Properties for Eye Contact

Property Name	Value
Landmarks	`NvAR_Point2f` array, which must be large enough to hold the number of points given by the product of the following: • `NvAR_Parameter_Config(BatchSize)` • `NvAR_Parameter_Config(Landmarks_Size)` To be allocated by the user.
HeadPose	Optional: `NvAR_Quaternion` array, which must be large enough to hold the number of quaternions equal to `NvAR_Parameter_Config(BatchSize)`. The OpenGL standards coordinate convention is used: When you look up from a camera, the co-ordinates are x (camera right), y (camera up), and z (toward the camera). To be allocated by the user.
LandmarksConfidence	Optional: An array of single-precision (32-bit) floating-point numbers, which must be large enough to hold the number of confidence values given by the product of the following: • `NvAR_Parameter_Config(BatchSize)` • `NvAR_Parameter_Config(LandmarksConfidence_Size` To be allocated by the user.
BoundingBoxes	Optional: `NvAR_BBoxes` structure that contains the detected face through face detection performed by the landmark detection feature. To be allocated by the user.
OutputGazeVector	Float array, which must be large enough to hold the two values (pitch and yaw) for the gaze angle in radians per image. For batch sizes larger than 1, it should hold `NvAR_Parameter_Config(BatchSize)` × 2 float values. To be allocated by the user.
OutputHeadTranslation	Optional: Float array, which must be large enough to hold the head translations (x,y,z) per image. For batch sizes larger than 1, it should hold `NvAR_Parameter_Config(BatchSize)` × 3 float values. To be allocated by the user.
GazeDirection	Optional: `NvAR_Point3f` array that is large enough to hold as many elements as `NvAR_Parameter_Config(BatchSize)`. Each element contains two `NvAR_Point3f` points. One point represents the center point (cx,cy,cz) between the eyes, and the other point represents a unit vector (ux, uy, uz) in the gaze direction for visualization. For batch sizes larger than 1, it should hold `NvAR_Parameter_Config(BatchSize)` × 2 `NvAR_Point3f` points. To be allocated by the user.

3.3.4. Body Detection Property Values

The following tables list the configuration, input, and output property values for Body Detection.

Table 3-10: Configuration Properties for Body Detection

Property Name	Value
FeatureDescription	String is free-form text that describes the feature. The string is set by the SDK and cannot be modified by the user.
CUDAStream	The CUDA stream, which is set by the user.
ModelDir	String that contains the path to the folder that contains the TensorRT package files. Set by the user.
ModelCacheDir	**(Windows on Arm only, such as RTX Spark)** Optional string path to the model runtime cache directory. Empty or unset selects `ModelDir/cache`. Set by the user before `NvAR_Load()`.
ModelCacheMode	**(Windows on Arm only, such as RTX Spark)** Optional unsigned integer that controls the model runtime cache: `0` = Auto (default), 1 = Disabled, 2 = ForceRegenerate. Set by the user before `NvAR_Load()`.
Temporal	Unsigned integer to enable (1) or disable (0) the temporal optimization of body detection. Set by the user.
FullBodyOnly	Unsigned integer to select the estimation mode: • 1: Full Body only (default). • 0: Full and upper body. Set by the user.

Table 3-11: Input Properties for Body Detection

Property Name	Value
Image	Interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. To be allocated and set by the user.

Table 3-12: Output Properties for Body Detection

Property Name	Value
BoundingBoxes	NvAR_BBoxes structure that holds the detected body boxes. To be allocated by the user.
BoundingBoxesConfidence	Optional: An array of single-precision (32-bit) floating-point numbers that contain the confidence values for each detected body box. To be allocated by the user.

3.3.5. Legacy 3D Body Pose Keypoint Tracking Property Values

The following tables list the configuration, input, and output property values for the legacy 3D Body Pose Keypoint Tracking feature.

Table 3-13: Configuration Properties for Legacy 3D Body Pose Keypoint Tracking

Property Name	Value
FeatureDescription	String that describes the feature.
CUDAStream	The CUDA stream. Set by the user.
ModelDir	String that contains the path to the folder that contains the TensorRT package files. Set by the user.
BatchSize	The number of inferences to be run at one time on the GPU. The maximum value is 1.
Mode	Unsigned integer that specifies the mode: High Performance (1) or High Quality (0). Default is 1. Set by the user.
UseCudaGraph	Boolean to enable (true) or disable (false) the use of CUDA Graphs for optimization. Set by the user.
Temporal	Unsigned integer to enable (1) or disable (0) the temporal optimization of Body Pose tracking. Set by the user.
NumKeyPoints	Unsigned integer that specifies the number of keypoints available, which is currently 34.
ReferencePose	`NvAR_Point3f` array that contains the reference pose for each of the keypoints. Set by the user.
FullBodyOnly	Unsigned integer to select the pose estimation mode: • Full Body only (1). Supports both high quality and high performance modes. • Full and upper body (0). Supports only high quality mode. The default is 1. Set by the user.
PostprocessJointAngle	Boolean to enable (true) or disable (false) the postprocessing steps for joint angles corresponding to the joints predicted with low confidence. To be used only when `FullBodyOnly` is set to 0. We recommend that you set this to true when input is upper-body image or video. The default is true. Set by the user.
TargetSeatedPoseForInterpolation	`NvAR_Quaternion` array that contains the target seated pose (for each of the 34 keypoints) to be used for post processing joint rotations. For the joints that are predicted with low confidence, the output pose will be interpolated to the corresponding pose specified in this target pose. This array is used when the SDK detects that the person in the input frame is in a seated pose. Used only when `FullBodyOnly` is set to 0.
TargetStandPoseForInterpolation	`NvAR_Quaternion` array that contains the target standing pose (for each of the 34 keypoints) to be used for post processing joint rotations.

Table 3-14: Input Properties for Legacy 3D Body Pose Keypoint Tracking

Property Name	Value
Image	Interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. To be allocated and set by the user.
FocalLength	Float value that specifies the focal length of the camera to be used for 3D Body Pose. The default value is 800.79041. To be allocated and set by the user.
Bounding-Boxes	Optional: `NvAR_BBoxes` structure that contains the number of bounding boxes that are equal to `BatchSize` on which to run 3D Body Pose detection. If not specified as an input property, body detection is automatically run on the input image. To be allocated by the user.

Table 3-15: Output Properties for Legacy 3D Body Pose Keypoint Tracking

Property Name	Value
Keypoints	`NvAR_Point2f` array, which must be large enough to hold the points given by the product of `NvAR_Parameter_Config(BatchSize)` and 34. To be allocated by the user.
Keypoints3D	`NvAR_Point3f` array, which must be large enough to hold the points given by the product of `NvAR_Parameter_Config(BatchSize)` and 34. To be allocated by the user.
JointAngles	`NvAR_Quaternion` array, which must be large enough to hold the joints given by the product of `NvAR_Parameter_Config(BatchSize)` and 34. Each value represents the local rotation of one joint relative to `Reference-Pose`. To be allocated by the user.
KeyPointsConfi-dence	An array of single-precision (32-bit) floating-point numbers, which must be large enough to hold the number of confidence values given by the product of `NvAR_Parameter_Config(BatchSize)` and 34. To be allocated by the user.
BoundingBoxes	`NvAR_BBoxes` structure that contains the detected body through body detection performed by the 3D Body Pose feature. To be allocated by the user.
TrackingBounding-Boxes	`NvAR_TrackingBBoxes` structure that contains the detected body through body detection that is completed by the 3D Body Pose feature and the tracking ID assigned by multi-person tracking. To be allocated by the user.

3.3.6. 3D Body Pose Estimation Property Values

The following tables list the configuration, input, and output properties for 3D Body Pose Estimation.

Configuration Properties for 3D Body Pose Estimation

Property Name	Value
FeatureDescription	String that describes the feature.
CUDAStream	CUDA stream used to run the feature. Set before `NvAR_Load`.
ModelDir	Path to the folder that contains the TensorRT package files. Set before `NvAR_Load`.
Mode	Unsigned integer mode selector. `NVAR_3DBODYPOSE_MODE_2D` is `0` and `NVAR_3DBODYPOSE_MODE_3D` is 1. The default is `NVAR_3DBODYPOSE_MODE_3D`.
MaxBoxesPerFrame	Read-only unsigned integer that reports the maximum number of tracked bodies that are accepted per frame.
NumKeyPoints	Read-only unsigned integer that reports the number of public keypoints per body. The current value is 77.
EnableContact	Unsigned integer that enables contact correction and contact-driven inverse kinematics (IK) in 3D mode. `0` disables it and 1 enables it. The default is 1. Set this property before `NvAR_Load`. Keep it enabled for fixed-camera scenes and whenever contact stability matters. Disable it when the camera moves, or when lower compute cost matters more than contact stabilization, because disabling it can increase visible drift and sliding.

Input Properties for 3D Body Pose Estimation

Property Name	Value
Image	Interleaved or chunky 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. Required for normal frame processing.
TrackingBoundingBoxes	`NvAR_TrackingBBoxes` structure that contains caller-provided tracked body boxes and tracking IDs for the current frame.
FocalLength	Optional focal length in pixels. If this property is unset or set to `0`, the feature derives a focal length from the input image size.
Flush	Unsigned integer stream-drain control. `0` is normal processing, and 1 flushes delayed outputs at the end of the stream.

Output Properties for 3D Body Pose Estimation

Property Name	Value
Ready	Read-only unsigned integer. 1 means that an output frame is available and that the registered output buffers contain valid data.
StreamFlushed	Read-only unsigned integer. 1 means that end-of-stream flushing is complete for the bound stream.
NumBodies	Read-only unsigned integer that reports the number of valid output bodies in the output frame that is ready.
TrackingBoundingBoxes	`NvAR_TrackingBBoxes` structure that receives the output-frame body boxes and tracking IDs.
KeyPoints	`NvAR_Point2f` array of 2D keypoints. Allocate at least `MaxBoxesPerFrame * NumKeyPoints` elements.
KeyPointsConfidence	Single-precision confidence array. Allocate at least `MaxBoxesPerFrame * NumKeyPoints` elements.
KeyPoints3D	`NvAR_Point3f` array of posed 3D keypoints in camera coordinates, in meters. Allocate at least `MaxBoxesPerFrame * NumKeyPoints` elements.
RestPose	`NvAR_Point3f` array of root-relative rest-pose keypoints in the SDK direct-rest basis, in meters. Allocate at least `MaxBoxesPerFrame * NumKeyPoints` elements.
RootPose	`NvAR_Transform3f` array of root pose values. Translation is in camera coordinates, in meters; rotation maps root-relative direct-rest points into the camera coordinate system. Allocate at least `MaxBoxesPerFrame` elements.
JointRotations	`NvAR_Quaternion` array of local joint rotations in SDK skeleton order and in the SDK direct-rest basis. Allocate at least `MaxBoxesPerFrame * NumKeyPoints` elements.
JointAngles	Compatibility alias for `JointRotations`.

3.3.7. Facial Expression Estimation Property Values

The following tables list the values for the configuration, input, and output properties for Facial Expression Estimation.

Table 3-16: Configuration Properties for Facial Expression Estimation

Property Name	Value
FeatureDescription	String that describes the feature. This property is read-only.
ModelDir	String that contains the path to the face model and the TensorRT package files. Set by the user.
ModelCacheDir	**(Windows on Arm only, such as RTX Spark)** Optional string path to the model runtime cache directory. Empty or unset selects `ModelDir/cache`. Set by the user before `NvAR_Load()`.
ModelCacheMode	**(Windows on Arm only, such as RTX Spark)** Optional unsigned integer that controls the model runtime cache: `0` = Auto (default), 1 = Disabled, 2 = ForceRegenerate. Set by the user before `NvAR_Load()`.
CUDAStream	Optional: The CUDA stream. Set by the user.
Temporal	Optional: Bitfield to control temporal filtering. • 0x001: Filter face detection. • 0x002: Filter facial landmarks. • 0x004: Filter rotational pose. • 0x010: Filter facial expressions. • 0x020: Filter gaze expressions. • 0x100: Enhance eye and mouth closure. Default is 0x037 (all on except 0x100). Set by the user.
Landmarks_Size	Unsigned integer, 68 or 126. Required array size of detected facial landmark points. Length of array must be 126, to accommodate {x,y} location of each of the detected points.
ExpressionCount	Unsigned integer. The number of expressions in the face model.
PoseMode	Specifies how to compute pose. 0 = 3DOF (default), 1 = 6DOF explicit. 6DOF is required for 3D translation output.
Mode	Flag to toggle landmark mode. Set 0 to enable Performance model for landmark detection. Set 1 to enable Quality model for landmark detection for higher accuracy. Default is 1.
EnableCheekPuff	(Experimental) Enables cheek puff blendshapes.

Table 3-17: Input Properties for Facial Expression Estimation

Property Name	Value
Landmarks	Optional: An `NvAR_Point2f` array that contains the landmark points of size `NvAR_Parameter_Config(Landmarks_Size)` that is returned by the landmark detection feature. If landmarks are not provided to this feature, an input image must be provided. To be allocated by the user.
Image	Optional: An interleaved (or chunky) 8-bit BGR input image in a CUDA buffer of type `NvCVImage`. If an input image is not provided as input, the landmark points must be provided to this feature as input. To be allocated by the user.
CameraIntrinsicParams	Optional: Camera intrinsic parameters. A three-element float array with elements corresponding to focal length, cx, and cy, respectively, of an ideal perspective camera. Any barrel or fisheye distortion should be removed or considered negligible. Used only if `PoseMode` is set to 1.

Table 3-18: Output Properties for Facial Expression Estimation

Property Name	Value
Landmarks	Optional: An `NvAR_Point2f` array, which must be large enough to hold the number of points of size `NvAR_Parameter_Config(Landmarks_Size)`.
Pose	Optional: `NvAR_Quaternion` Pose rotation quaternion. Coordinate frame is NvAR Camera 3D pace. To be allocated by the user.
PoseTranslation	Optional: `NvAR_Point3f` Pose 3D Translation. Computed only if `PoseMode` = 1. Translation coordinates are in NvAR Camera 3D Space coordinates, in which the units are centimeters. To be allocated by the user.
LandmarksConfidence	Optional: An array of single-precision (32-bit) floating-point numbers, which must be large enough to hold the number of confidence values of size `NvAR_Parameter_Config(LandmarksConfidence_Size)`. To be allocated by the user.
BoundingBoxes	Optional: `NvAR_BBoxes` structure that contains the detected face that is determined internally. To be allocated by the user.
BoundingBoxesConfidence	Optional: An array of single-precision (32-bit) floating-point numbers, which must be large enough to hold the number of confidence values of size `NvAR_Parameter_Config(BoundingBoxesConfidence_Size)`. To be allocated by the user.
ExpressionCoefficients	The array into which the expression coefficients will be placed, if desired. Query `ExpressionCount` to determine the size for this array. To be allocated by the user. The corresponding expression shapes are in the following order: `BrowDown_L`, `BrowDown_R`, `BrowInnerUp_L`, `BrowInnerUp_R`, `BrowOuterUp_L`, `BrowOuterUp_R`, `CheekPuff_L`, `CheekPuff_R`, `CheekSquint_L`, `CheekSquint_R`, `EyeBlink_L`, `EyeBlink_R`, `EyeLookDown_L`, `EyeLookDown_R`, `EyeLookIn_L`, `EyeLookIn_R`, `EyeLookOut_L`, `EyeLookOut_R`, `EyeLookUp_L`, `EyeLookUp_R`, `EyeSquint_L`, `EyeSquint_R`, `EyeWide_L`, `EyeWide_R`, `JawForward`, `JawLeft`, `JawOpen`, `JawRight`, `MouthClose`, `MouthDimple_L`, `MouthDimple_R`, `MouthFrown_L`, `MouthFrown_R`, `MouthFunnel`, `MouthLeft`, `MouthLowerDown_L`, `MouthLowerDown_R`, `MouthPress_L`, `MouthPress_R`, `MouthPucker`, `MouthRight`, `MouthRollLower`, `MouthRollUpper`, `MouthShrugLower`, `MouthShrugUpper`, `MouthSmile_L`, `MouthSmile_R`, `MouthStretch_L`, `MouthStretch_R`, `MouthUpperUp_L`, `MouthUpperUp_R`, `NoseSneer_L`, `NoseSneer_R`

3.3.8. LipSync Property Values

The following tables list the values for the configuration, input, and output properties for LipSync.

Table 3-19: Configuration Properties for LipSync

Property Name	Value
Feature-Description	String that describes the feature. This property is read-only.
ModelDir	String that contains the path to the face model and the TensorRT package files. Set by the user.
Language	Unsigned 32-bit value that selects which language-specific LipSync TensorRT model is loaded from `ModelDir`. `0`: Generic (default). 1: German (`lipsync_DE` variant). 2: French (`lipsync_FR` variant). 3: Spanish (`lipsync_ES` variant). Set this parameter before calling `NvAR_Load`. The deployment must include the matching model package for the selected language. Set by the user.
CUDAStream	Optional: The CUDA stream. Set by the user.
VideoFPS	The video frame rate in frames per second. Set by the user.
SampleRate	The sample rate for the audio input. The feature supports 16-kHz audio only. This property is read-only.
NumChannels	The number of channels for the audio input. The feature supports mono channel audio only. This property is read-only.
NumInitial-Frames	The number of initial audio frames before the first image can be generated. You need to provide `NumInitialFrames` number of video frames before the first output is available at `Image` output. The default value is 14. This property is read-only.

Table 3-20: Input Properties for LipSync

Property Name	Value
Image	Chunky/packed CUDA buffer. Supported formats: BGR-U8, BGR-U16, BGRA-U16, RGBA-U16. The component type (U8 or U16) must match between the input and output images. Requirements: • Resolution between 360p and 4K. Face region in the input frame up to 512×512 resolution. • The full face of the subject is visible. • Maximum of ±30 degrees head movement in each axis. • Moderate to good lighting conditions. • Clear face features; no occlusion.
AudioFrameBuffer	Raw audio frame buffer in CPU ranging from –1.0 to 1.0, inclusive. When the LipSync feature is run, it assumes that the contents of the audio frame buffer are synchronized with the current input video frame. The length of the audio frame should approximately match the duration of the video frame. The caller can vary the length of each audio frame to maintain synchronization. Audio requirements: • Sample rate: 16 kHz. • Channel: 1 (mono). • Only one person speaking at a time. • No background noise.
LipSyncRegionData	Optional: An `NvAR_LipSyncRegionData` object that specifies the regions and per-region settings for LipSync, including tracking ID, bypass, region type, and speaking flag. When not set (nullptr), the feature operates in single full-frame region mode. **Attention** `SpeakerData` is no longer supported. Use `LipSyncRegionData` instead.
HeadMovementSpeed	Optional: Speed of head movement in the input video. • 0: Static/slow-moving head. • 1: Fast-moving head. Default: 0. Set by the user.

Table 3-21: Output Properties for LipSync

Property Name	Value
Image	Chunky/packed CUDA buffer. Supported formats: BGR-U8, BGR-U16, BGRA-U16, RGBA-U16. The component type must match the input image.
Ready	Flag that is set to a non-zero value when the first output video frame is generated.
Activation	An `NvAR_LipSyncActivation` structure that provides the activation strength and the location and size of the animated face. The `strength` field is a value in the range 0–1 that indicates the activation level. When 0, the original face was copied directly to the output without modification. When 1, the original face was completely replaced by the animated face. The `center_x`, `center_y`, and `size` fields provide the face center location and size in pixels.

3.3.9. Active Speaker Detection Property Values

The following tables list the values for the configuration, input, and output properties for Active Speaker Detection.

Table 3-22: Configuration Properties for Active Speaker Detection

Property Name	Value
FeatureDescription	String that describes the feature. This property is read-only.
ModelDir	String that contains the path to the TensorRT model files. Set by the user.
CUDAStream	Optional: The CUDA stream. Set by the user.
VideoFPS	Video frames per second. Must be in the range (0.0, 120.0]. Default: 30.0.
NumAudioStreams	Number of audio tracks to process. Must be in the range [1, 30]. Default: 1.
SampleRate	Audio sample rate in Hz. Must be in the range (0, 96000]. Default: 44100.
SyncTolerance	Minimum audio-visual sync score in the range (0, 1) to consider a face as speaking. Higher values require stronger sync evidence. Set by the user.
MaxNumOutputIdentities	Maximum number of output identities that can be tracked simultaneously. This property is read-only. Default: 30.
MaxSyncFaces	Maximum number of tracked faces to evaluate with sync discrimination on each firing. `0` disables the limit. This property does not limit the output face count. It is not a hard cap on established speakers over time. Default: 0.
Flags	Flag bits for shot-change handling and active-audio filtering behavior. Set by the user.

The following `Flags` are allowed:

- `NVARACTIVESPEAKERDETECTION_FLAG_SHOT_CHANGED`: Indicates that the frame is a shot change.
- `NVARACTIVESPEAKERDETECTION_FLAG_DETECT_SHOT_CHANGE`: Enables automatic shot-change detection.
- `NVARACTIVESPEAKERDETECTION_FLAG_FILTER_SILENT_TRACKS`: Filters silent audio tracks out of the active audio list.
- `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VAD`: Runs denoiser voice-activity detection on the supplied audio tracks and filters inactive tracks.
- `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VA_SMOOTHING_LOW`: Enables low voice-activity smoothing.
- `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VA_SMOOTHING_HIGH`: Enables high voice-activity smoothing.

`NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VAD` and `NVARACTIVESPEAKERDETECTION_FLAG_FILTER_SILENT_TRACKS` are mutually exclusive. Use at most one voice-activity smoothing flag at a time.

When no active-audio filtering flag is set, `ActiveAudioIDs` is treated as authoritative, such as with diarization-driven input. When denoiser VAD or silent-track filtering is enabled, `ActiveAudioIDs` is the candidate audio list that can be filtered by the SDK.

The `NewShot` input remains supported for backward compatibility. If both `NewShot` and `Flags` are set, `NewShot` takes precedence for shot-change behavior.

Table 3-23: Input Properties for Active Speaker Detection

Property Name	Value
Image	Input image in GPU memory with `NVCV_BGR`, `NVCV_U8` format. Maximum supported resolution is 4096 × 2160. Set by the user.
AudioFrameData	Input audio frame data containing audio samples for all tracks. Type: *NvAR_AudioFrameData*. The audio data should be synchronized with the input video frame and contain floating-point samples in the range [-1.0, 1.0]. Set by the user.
ActiveAudioIDs	Array of active audio track IDs. Type: *NvAR_ActiveAudioIds*. Without VAD or silent-track filtering, only listed tracks are processed. With filtering enabled, this is the candidate list that can be filtered by the SDK. Sample applications can derive this list from diarization JSON. Set by the user.
NewShot	Shot-change mode for the current stream: • 0: No shot change. • 1: Shot change occurred. • 255: Auto-detect shot changes (default). Set by the user.
Flush	Flush mode for the current stream: • 0: Normal processing. • 1: Flush mode (finalize outputs). Set by the user.

Table 3-24: Output Properties for Active Speaker Detection

Property Name	Value
ActiveSpeakerTrackingData	Output tracking data containing detected faces with speaker information. Type: *NvAR_ActiveSpeakerTrackingData*. Each tracked face includes the following: • Bounding box coordinates. • Unique tracking ID. • Associated audio track ID. • Detection confidence score. • Speaking status flag. To be allocated by the user.
Ready	Output ready status for the current: • 0: Output not ready. • 1: Output ready and valid. The feature requires multiple frames to accumulate before producing valid output.

3.4. Using the AR Features

This section provides information about how to use the AR features.

3.4.1. Face Detection and Tracking

This section provides information about how to use the Face Detection and Tracking feature.

3.4.1.1 Face Detection for Static Frames (Images)

To obtain detected bounding boxes, you can explicitly instantiate and run the face detection feature as follows, with the feature taking an image buffer as input.

The following example runs the Face Detection AR feature with an input image buffer and output memory to hold bounding boxes:

```
//Set input image buffer
NvAR_SetObject(faceDetectHandle, NvAR_Parameter_Input(Image), &
↪inputImageBuffer, sizeof(NvCVImage));

//Set output memory for bounding boxes
NvAR_BBoxes = output_boxes{};
output_bboxes.boxes = new NvAR_Rect[25];
output_bboxes.max_boxes = 25;
NvAR_SetObject(faceDetectHandle, NvAR_Parameter_Output(BoundingBoxes), &
↪output_bboxes, sizeof(NvAR_BBoxes));

//Optional: If desired, set memory for bounding-box confidence values
NvAR_Run(faceDetectHandle);
```

3.4.1.2 Face Tracking for Temporal Frames (Videos)

If Temporal is enabled, such as when you process a video frame instead of an image, only one face is returned. The largest face appears for the first frame, and this face is subsequently tracked over the following frames.

However, explicitly calling the face detection feature is not the only way to obtain a bounding box that denotes detected faces. For more information about how to use the Landmark Detection AR feature and return a face bounding box, see *Landmark Detection and Tracking*.

3.4.2. Landmark Detection and Tracking

This section provides information about how to use the Landmark Detection and Tracking feature.

3.4.2.1 Landmark Detection for Static Frames (Images)

Typically, the input to the landmark detection feature is an input image and a batch of bounding boxes. Currently, the maximum value is 1. These boxes denote the regions of the image that contain the faces on which you want to run landmark detection.

The following example runs the Landmark Detection AR feature after obtaining bounding boxes from Face Detection:

```
//Set input image buffer
NvAR_SetObject(landmarkDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Pass output bounding boxes from face detection as an input on which
//landmark detection is to be run
NvAR_SetObject(landmarkDetectHandle,
  NvAR_Parameter_Input(BoundingBoxes), &output_bboxes,
  sizeof(NvAR_BBoxes));

//Set landmark detection mode: Performance (0; default) or Quality (1)
unsigned int mode = 0; // Choose performance mode
NvAR_SetU32(landmarkDetectHandle, NvAR_Parameter_Config(Mode), mode);

//Set output buffer to hold detected facial keypoints
std::vector<NvAR_Point2f> facial_landmarks;
facial_landmarks.assign(OUTPUT_SIZE_KPTS, {0.f, 0.f});
NvAR_SetObject(landmarkDetectHandle, NvAR_Parameter_Output(Landmarks),
  facial_landmarks.data(),sizeof(NvAR_Point2f));

NvAR_Run(landmarkDetectHandle);
```

3.4.2.2 Alternative Usage of Landmark Detection

As described in *Configuration Properties for Landmark Tracking*, the Landmark Detection AR feature supports some optional parameters that determine how the feature can be run.

If bounding boxes are not provided to the Landmark Detection AR feature as inputs, face detection is automatically run on the input image, and the largest face bounding box is selected on which to run landmark detection.

If BoundingBoxes is set as an output property, the property is populated with the selected bounding box that contains the face on which the landmark detection was run. Landmarks is not an optional property; to explicitly run this feature, this property must be set with a provided output buffer.

3.4.2.3 Landmark Tracking for Temporal Frames (Videos)

Additionally, if Temporal is enabled, such as when you process a video stream and face detection is run explicitly, only one bounding box is supported as an input for landmark detection.

When face detection is not explicitly run, by providing an input image instead of a bounding box, the largest detected face is automatically selected. The detected face and landmarks are then tracked as an optimization across temporally related frames.

Note

The internally determined bounding box can be queried from this feature, but is not required for the feature to run.

The following example uses the Landmark Detection AR feature to obtain landmarks directly from the image, without first explicitly running Face Detection:

```
//Set input image buffer
NvAR_SetObject(landmarkDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Set output memory for landmarks
std::vector<NvAR_Point2f> facial_landmarks;
facial_landmarks.assign(batchSize * OUTPUT_SIZE_KPTS, {0.f, 0.f});
NvAR_SetObject(landmarkDetectHandle, NvAR_Parameter_Output(Landmarks),
  facial_landmarks.data(),sizeof(NvAR_Point2f));

//Set landmark detection mode: Performance (0; default) or Quality (1)
unsigned int mode = 0; // Choose performance mode
NvAR_SetU32(landmarkDetectHandle, NvAR_Parameter_Config(Mode), mode);

//Optional: If desired, set memory for bounding box
NvAr_BBoxes = output_boxes{};
output_bboxes.boxes = new NvAR_Rect[25];
output_bboxes.max_boxes = 25;
NvAR_SetObject(landmarkDetectHandle,
  NvAR_Parameter_Output(BoundingBoxes), &output_bboxes,
  sizeof(NvAr_BBoxes));

//Optional: If desired, set memory for pose, landmark confidence, or
//even bounding box confidence

NvAR_Run(landmarkDetectHandle);
```

3.4.3. Eye Contact

This feature estimates the gaze of a person from an eye patch that was extracted by using landmarks and redirects the eyes to make the person look at the camera in a permissible range of eye and head

angles. The feature also supports a mode where the estimation can be obtained without redirection. The eye contact feature can be invoked by using the `GazeRedirection` feature ID.

Eye contact feature has the following options:

- Gaze Estimation
- Gaze Redirection

In this release, gaze estimation and redirection of only one face in the frame is supported.

3.4.3.1 Gaze Estimation

The estimation of gaze requires face detection and landmarks as input. The inputs to the gaze estimator are an input image buffer and buffers to hold facial landmarks and confidence scores. The output of gaze estimation is the gaze vector (pitch, yaw) values in radians. A float array must be set as the output buffer to hold estimated gaze. The `GazeRedirect` parameter must be set to false.

The following example runs the Gaze Estimation with an input image buffer and output memory to hold the estimated gaze vector:

```
bool bGazeRedirect=false;
NvAR_SetU32(gazeRedirectHandle, NvAR_Parameter_Config(GazeRedirect),
  bGazeRedirect);

//Set input image buffer
NvAR_SetObject(gazeRedirectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Set output memory for gaze vector
float gaze_angles_vector[2];
NvvAR_SetF32Array(gazeRedirectHandle,
  NvAR_Parameter_Output(OutputGazeVector), gaze_angles_vector, batchSize
  * 2);

//Optional: Set output memory for landmarks, head pose, head
//translation, and gaze direction
std::vector<NvAR_Point2f> facial_landmarks;
facial_landmarks.assign(batchSize * OUTPUT_SIZE_KPTS, {0.f, 0.f});
NvAR_SetObject(gazeRedirectHandle, NvAR_Parameter_Output(Landmarks),
  facial_landmarks.data(),sizeof(NvAR_Point2f));

NvAR_Quaternion head_pose;
NvAR_SetObject(gazeRedirectHandle, NvAR_Parameter_Output(HeadPose),
  &head_pose, sizeof(NvAR_Quaternion));

float head_translation[3] = {0.f};
NvAR_SetF32Array(gazeRedirectHandle,
  NvAR_Parameter_Output(OutputHeadTranslation), head_translation,
  batchSize * 3);

NvAR_Run(gazeRedirectHandle);
```

3.4.3.2 Gaze Redirection

Gaze Redirection takes identical inputs as the gaze estimation. In addition to the outputs of gaze estimation, to store the gaze redirected image, an output image buffer of the same size as the input image buffer must be set. The gaze is redirected to look at the camera within a certain range of gaze angles and head poses. Outside this range, the feature disengages. Head pose, head translation, and gaze direction can be optionally set as outputs. The `GazeRedirect` parameter must be set to true.

The following example runs Gaze Redirection with an input image buffer and output memory to hold the estimated gaze vector and an output image buffer to hold the gaze redirected image.

```
bool bGazeRedirect=true;
NvAR_SetU32(gazeRedirectHandle, NvAR_Parameter_Config(GazeRedirect),
  bGazeRedirect);

//Set input image buffer
NvAR_SetObject(gazeRedirectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Set output memory for gaze vector
float gaze_angles_vector[2];
NvvAR_SetF32Array(gazeRedirectHandle,
  NvAR_Parameter_Output(OutputGazeVector), gaze_angles_vector, batchSize
  * 2);

//Set output image buffer
NvAR_SetObject(gazeRedirectHandle, NvAR_Parameter_Output(Image),
  &outputImageBuffer, sizeof(NvCVImage));

NvAR_Run(gazeRedirectHandle);
```

3.4.3.3 Randomized Look Away

A continuous redirection of gaze to look at the camera might give a perception of “stare.” Some users might find this effect unnatural or undesired. In order to occasionally break eye contact, we provide randomized look aways in gaze redirection which can be optionally enabled. While the gaze is always expected to redirect toward the camera within the range of operation, enabling look away will make the user occasionally break gaze lock to the camera with a micro-movement of the eyes at randomly chosen time intervals. The `EnableLookAway` parameter must be set to true to enable this feature. Additionally, parameters `LookAwayOffsetMax`, `LookAwayIntervalMin`, and `LookAwayIntervalRange` are optional parameters that can be used to tune the extent and frequency of look away. For a detailed description and default settings of these parameters, see *Configuration Properties for Eye Contact*.

3.4.3.4 Range Control

The gaze redirection feature redirects the eyes to look at the camera within a certain range of head and eye motion in which eye contact is desired and looks natural. Beyond this range, the feature gradually transitions away from looking at the camera toward the estimated gaze and eventually turns off in a seamless manner. To provide for various use cases and user preferences, we provide range parameters for the user to control the range of gaze angles and head poses in which gaze redirection occurs and the range in which transition occurs before the redirection is turned off. These are optional parameters.

`GazePitchThresholdLow` and `GazeYawThresholdLow` define the parameters for the pitch and yaw angles of the estimated gaze within which gaze is redirected toward the camera. Beyond these angles,

redirected gaze transitions away from the camera and toward the estimated gaze, turning off redirection beyond `GazePitchThresholdHigh` and `GazeYawThresholdHigh`, respectively. Similarly, for head pose, `HeadPitchThresholdLow` and `HeadYawThresholdLow` define the parameters for pitch and yaw angles of the head pose within which gaze is redirected toward the camera. Beyond these angles, redirected gaze transitions away from the camera and toward the estimated gaze, turning off redirection beyond `HeadPitchThresholdHigh` and `HeadYawThresholdHigh`. For a detailed description and default settings of these parameters, see *Configuration Properties for Eye Contact*.

3.4.3.5 Multi-Person Support

Set the `MultiPerson` configuration parameter to `1` to redirect the gaze of every detected face, up to eight faces per video stream. For a detailed description and default setting of this parameter, refer to *Configuration Properties for Eye Contact*.

3.4.4. Legacy 3D Body Pose Tracking

3D Body Pose Tracking is a legacy feature. For new applications, use *3D Body Pose Estimation*.

This feature relies on temporal information to track the person in the scene, where the keypoints information from the previous frame is used to estimate the keypoints of the next frame.

3D Body Pose Tracking consists of the following parts:

- Body Detection
- 3D Keypoint Detection

The feature supports single or multiple people in the frame and both full or upper-body images and videos.

3.4.4.1 3D Body Pose Tracking for Static Frames (Images)

You can obtain the bounding boxes that encapsulate the people in the scene. To obtain detected bounding boxes, you can explicitly instantiate and run body detection and pass the image buffer as input.

The following example runs the Body Detection with an input image buffer and output memory to hold bounding boxes:

```
//Set input image buffer
NvAR_SetObject(bodyDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Set output memory for bounding boxes
NvAR_BBoxes = output_boxes{};
output_bboxes.boxes = new NvAR_Rect[25];
output_bboxes.max_boxes = 25;
NvAR_SetObject(bodyDetectHandle, NvAR_Parameter_Output(BoundingBoxes),
  &output_bboxes, sizeof(NvAR_BBoxes));

//Optional: If desired, set memory for bounding-box confidence values

NvAR_Run(bodyDetectHandle);
```

The input to 3D Body Keypoint Detection is an input image. It outputs the 2D keypoints, 3D keypoints, keypoint confidence scores, and bounding box encapsulating the person.

The following example runs the 3D Body Pose Detection AR feature:

```
//Set input image buffer
NvAR_SetObject(keypointDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Pass output bounding boxes from body detection as an input on which
//landmark detection is to be run
NvAR_SetObject(keypointDetectHandle,
  NvAR_Parameter_Input(BoundingBoxes), &output_bboxes,
  sizeof(NvAR_BBoxes));

//Set output buffer to hold detected keypoints
std::vector<NvAR_Point2f> keypoints;
std::vector<NvAR_Point3f> keypoints3D;
std::vector<NvAR_Quaternion> jointAngles;
std::vector<float> keypoints_confidence;

// Get the number of keypoints
unsigned int numKeyPoints;

NvAR_GetU32(keyPointDetectHandle, NvAR_Parameter_
↪Config(NumKeyPoints),
  &numKeyPoints);
keypoints.assign(batchSize * numKeyPoints , {0.f, 0.f});
keypoints3D.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f});
jointAngles.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f, 0.f});
NvAR_SetObject(keyPointDetectHandle, NvAR_Parameter_
↪Output(KeyPoints),
  keypoints.data(), sizeof(NvAR_Point2f));
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPoints3D), keypoints3D.data(),
  sizeof(NvAR_Point3f));
NvAR_SetF32Array(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPointsConfidence),
  keypoints_confidence.data(), batchSize * numKeyPoints);
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(JointAngles), jointAngles.data(),
  sizeof(NvAR_Quaternion));

//Set output memory for bounding boxes
NvAR_BBoxes = output_boxes{};
output_bboxes.boxes = new NvAR_Rect[25];
output_bboxes.max_boxes = 25;
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(BoundingBoxes), &output_bboxes,
  sizeof(NvAR_BBoxes));

NvAR_Run(keyPointDetectHandle);
```

3.4.4.2 3D Body Pose Tracking for Temporal Frames (Videos)

The feature relies on temporal information to track the person in the scene. The keypoints information from the previous frame is used to estimate the keypoints of the next frame.

The following example uses the 3D Body Pose Tracking AR feature to obtain 3D Body Pose Keypoints directly from the image:

```
//Set input image buffer
NvAR_SetObject(keypointDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Pass output bounding boxes from body detection as an input on which
//landmark detection is to be run
NvAR_SetObject(keypointDetectHandle,
  NvAR_Parameter_Input(BoundingBoxes), &output_bboxes,
  sizeof(NvAR_BBoxes));

//Set output buffer to hold detected keypoints
std::vector<NvAR_Point2f> keypoints;
std::vector<NvAR_Point3f> keypoints3D;
std::vector<NvAR_Quaternion> jointAngles;
std::vector<float> keypoints_confidence;

// Get the number of keypoints
unsigned int numKeyPoints;
NvAR_GetU32(keyPointDetectHandle, NvAR_Parameter_Config(NumKeyPoints),
  &numKeyPoints);
keypoints.assign(batchSize * numKeyPoints , {0.f, 0.f});
keypoints3D.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f});
jointAngles.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f, 0.f});
NvAR_SetObject(keyPointDetectHandle, NvAR_Parameter_Output(KeyPoints),
  keypoints.data(), sizeof(NvAR_Point2f));
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPoints3D), keypoints3D.data(),
  sizeof(NvAR_Point3f));
NvAR_SetF32Array(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPointsConfidence),
  keypoints_confidence.data(), batchSize * numKeyPoints);
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(JointAngles), jointAngles.data(),
  sizeof(NvAR_Quaternion));

//Set output memory for bounding boxes
NvAR_BBoxes = output_boxes{};
output_bboxes.boxes = new NvAR_Rect[25];
output_bboxes.max_boxes = 25;
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(BoundingBoxes), &output_bboxes,
  sizeof(NvAR_BBoxes));

NvAR_Run(keyPointDetectHandle);
```

3.4.4.3 Multi-Person Tracking for 3D Body Pose Tracking

The feature provides the ability to track multiple people in the following ways:

- In the scene across different frames.
- When they leave the scene and enter the scene again.
- When they are completely occluded by an object or another person and reappear (controlled using Shadow Tracking Age).

Shadow Tracking Age is a parameter that represents the period of time where a target is still being tracked in the background even when the target is not associated with a detector object. When a target is not associated with a detector object for a time frame, `shadowTrackingAge`, an internal variable of the target, is incremented. After the target is associated with a detector object, `shadowTrackingAge` is reset to zero. When the target age reaches the shadow tracking age, the target is discarded and is no longer tracked. This is measured by the number of frames; the default is 90.

Probation Age is the length of probationary period. After an object reaches this age, it is considered to be valid and is appointed an ID. This helps with false positives, where false objects are detected for only a few frames. This is measured by the number of frames; the default is 10.

Maximum Targets Tracked is the maximum number of targets to be tracked, which can be composed of the targets that are active in the frame and ones in shadow-tracking mode. When you select this value, keep the active and inactive targets in mind. The minimum is 1 and the default is 30.

Note

Currently, we actively track only eight people in the scene. More than eight people can appear throughout the video, but only a maximum of eight people in a given frame. Temporal mode is not supported for Multi-Person Tracking. The batch size should be 8 when Multi-Person Tracking is enabled.

The following example uses the 3D Body Pose Tracking AR feature to enable multi-person tracking and obtain the tracking ID for each person:

```
// Set input image buffer
NvAR_SetObject(keypointDetectHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

// Enable Multi-Person Tracking
NvAR_SetU32(keyPointDetectHandle, NvAR_Parameter_Config(TrackPeople),
  bEnablePeopleTracking);

// Set Shadow Tracking Age
NvAR_SetU32(keyPointDetectHandle,
  NvAR_Parameter_Config(ShadowTrackingAge), shadowTrackingAge);

// Set Probation Age
NvAR_SetU32(keyPointDetectHandle, NvAR_Parameter_
↪Config(ProbationAge),
  probationAge);

// Set Maximum Targets to be tracked
NvAR_SetU32(keyPointDetectHandle,
  NvAR_Parameter_Config(MaxTargetsTracked), maxTargetsTracked);
```

(continues on next page)

(continued from previous page)

```
// Set output buffer to hold detected keypoints
std::vector<NvAR_Point2f> keypoints;
std::vector<NvAR_Point3f> keypoints3D;
std::vector<NvAR_Quaternion> jointAngles;
std::vector<float> keypoints_confidence;

// Get the number of keypoints
unsigned int numKeyPoints;
NvAR_GetU32(keyPointDetectHandle, NvAR_Parameter_
↪Config(NumKeyPoints),
  &numKeyPoints);
keypoints.assign(batchSize * numKeyPoints , {0.f, 0.f});
keypoints3D.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f});
jointAngles.assign(batchSize * numKeyPoints , {0.f, 0.f, 0.f, 0.f});
NvAR_SetObject(keyPointDetectHandle, NvAR_Parameter_
↪Output(KeyPoints),
  keypoints.data(), sizeof(NvAR_Point2f));
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPoints3D), keypoints3D.data(),
  sizeof(NvAR_Point3f));
NvAR_SetF32Array(keyPointDetectHandle,
  NvAR_Parameter_Output(KeyPointsConfidence),
  keypoints_confidence.data(), batchSize * numKeyPoints);
NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(JointAngles), jointAngles.data(),
  sizeof(NvAR_Quaternion));

// Set output memory for tracking bounding boxes
NvAR_TrackingBBoxes output_tracking_bboxes{};
std::vector<NvAR_TrackingBBox> output_tracking_bbox_data;
output_tracking_bbox_data.assign(maxTargetsTracked, { 0.f, 0.f, 0.f,
  0.f, 0 });
output_tracking_bboxes.boxes = output_tracking_bbox_data.data();
output_tracking_bboxes.max_boxes = (uint8_t)output_tracking_bbox_
↪size;

NvAR_SetObject(keyPointDetectHandle,
  NvAR_Parameter_Output(TrackingBoundingBoxes), &output_tracking_
↪bboxes,
  sizeof(NvAR_TrackingBBoxes));

NvAR_Run(keyPointDetectHandle);
```

3.4.5. 3D Body Pose Estimation

3D Body Pose Estimation predicts body pose for tracked people in video frames. Create the feature with `NvAR_Feature_3DBodyPoseEstimation` from `nvAR3DBodyPoseEstimation.h`.

The feature expects caller-provided tracked body boxes, and each input box must include a stable tracking ID in `NvAR_TrackingBBoxes`. If your application needs body detection, run Body Detection before 3D Body Pose Estimation and pass its tracked boxes to this feature.

Note

Pose and depth estimates can be less accurate when a person is only partially visible or is mostly outside the frame, because the feature has fewer observations of the body. In multi-person scenes, heavy overlap or ambiguous tracking boxes can cause temporary identity swaps. Because 3D inference processes temporal chunks, unstable boxes or identity-swapped boxes can affect pose estimates across the chunk. Stable tracking IDs from the upstream tracker are important for preserving per-person temporal state and accurate 3D pose estimation.

The feature supports the following modes:

- `NVAR_3DBODYPOSE_MODE_2D`: returns 2D body keypoints, keypoint confidence values, output tracking boxes, `Ready`, `NumBodies`, and `StreamFlushed`.
- `NVAR_3DBODYPOSE_MODE_3D`: returns the 2D outputs, and also returns 3D keypoints in camera coordinates, rest pose, root pose, and joint rotations. The 3D output is delayed, so use `Ready` to determine when an output frame is available.

Contact correction and contact-driven inverse kinematics (IK) are enabled by default. Keep `NvAR_Parameter_Config(EnableContact)` enabled for fixed-camera scenes and whenever contact stability matters. Disable it before `NvAR_Load` when the camera moves, or when lower compute cost matters more than contact stabilization, because disabling it can increase visible drift and sliding.

When the stream ends, set `NvAR_Parameter_Input(Flush)` to 1 and keep calling `NvAR_Run` until `NvAR_Parameter_Output(StreamFlushed)` returns 1.

The following example shows the core 3D usage pattern:

```
NvAR_FeatureHandle bodyPoseHandle{};
NvAR_Create(NvAR_Feature_3DBodyPoseEstimation, &bodyPoseHandle);

NvAR_SetString(bodyPoseHandle, NvAR_Parameter_Config(ModelDir),
  modelDir);
NvAR_SetCudaStream(bodyPoseHandle, NvAR_Parameter_Config(CUDAStream),
  stream);
NvAR_SetU32(bodyPoseHandle, NvAR_Parameter_Config(Mode),
  NVAR_3DBODYPOSE_MODE_3D);
NvAR_Load(bodyPoseHandle);

uint32_t maxBodies{};
uint32_t numKeyPoints{};
NvAR_GetU32(bodyPoseHandle, NvAR_Parameter_Config(MaxBoxesPerFrame),
  &maxBodies);
NvAR_GetU32(bodyPoseHandle, NvAR_Parameter_Config(NumKeyPoints),
  &numKeyPoints);

std::vector<NvAR_TrackingBBox> inputBoxStorage(maxBodies);
NvAR_TrackingBBoxes inputBoxes{};
inputBoxes.boxes = inputBoxStorage.data();
inputBoxes.max_boxes = static_cast<uint8_t>(maxBodies);

std::vector<NvAR_TrackingBBox> outputBoxStorage(maxBodies);
NvAR_TrackingBBoxes outputBoxes{};
outputBoxes.boxes = outputBoxStorage.data();
outputBoxes.max_boxes = static_cast<uint8_t>(maxBodies);
```

(continues on next page)

(continued from previous page)

```
std::vector<NvAR_Point2f> keypoints(maxBodies * numKeyPoints);
std::vector<float> confidence(maxBodies * numKeyPoints);
std::vector<NvAR_Point3f> keypoints3D(maxBodies * numKeyPoints);
std::vector<NvAR_Point3f> restPose(maxBodies * numKeyPoints);
std::vector<NvAR_Transform3f> rootPose(maxBodies);
std::vector<NvAR_Quaternion> jointRotations(maxBodies * numKeyPoints);

NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));
NvAR_SetObject(bodyPoseHandle,
  NvAR_Parameter_Input(TrackingBoundingBoxes), &inputBoxes,
  sizeof(NvAR_TrackingBBoxes));
NvAR_SetF32(bodyPoseHandle, NvAR_Parameter_Input(FocalLength),
  focalLength);

NvAR_SetObject(bodyPoseHandle,
  NvAR_Parameter_Output(TrackingBoundingBoxes), &outputBoxes,
  sizeof(NvAR_TrackingBBoxes));
NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Output(KeyPoints),
  keypoints.data(), sizeof(NvAR_Point2f));
NvAR_SetF32Array(bodyPoseHandle,
  NvAR_Parameter_Output(KeyPointsConfidence), confidence.data(),
  static_cast<int32_t>(confidence.size()));
NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Output(KeyPoints3D),
  keypoints3D.data(), sizeof(NvAR_Point3f));
NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Output(RestPose),
  restPose.data(), sizeof(NvAR_Point3f));
NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Output(RootPose),
  rootPose.data(), sizeof(NvAR_Transform3f));
NvAR_SetObject(bodyPoseHandle, NvAR_Parameter_Output(JointRotations),
  jointRotations.data(), sizeof(NvAR_Quaternion));

// For each input frame, update inputBoxes.boxes and inputBoxes.num_boxes.
NvAR_SetU32(bodyPoseHandle, NvAR_Parameter_Input(Flush), 0);
NvAR_Run(bodyPoseHandle);

uint32_t ready{};
NvAR_GetU32(bodyPoseHandle, NvAR_Parameter_Output(Ready), &ready);
if (ready != 0) {
  uint32_t numBodies{};
  NvAR_GetU32(bodyPoseHandle, NvAR_Parameter_Output(NumBodies),
    &numBodies);
  // outputBoxes.boxes[0..numBodies) contains the output tracking IDs.
  // The registered keypoint, rest-pose, root-pose, and rotation buffers
  // are valid for the same delayed output frame.
}
```

3.4.6. Facial Expression Estimation

This section provides information about how to use the Facial Expression Estimation feature.

3.4.6.1 Facial Expression Estimation for Static Frames (Images)

Typically, the input to the Facial Expression Estimation feature is an input image and a set of detected landmark points that correspond to the face on which we want to estimate face expression coefficients.

The following example shows the typical usage of this feature, where the detected facial keypoints from the Landmark Detection feature are passed as input to this feature:

```
//Set facial keypoints from Landmark Detection as an input
err = NvAR_SetObject(faceExpressionHandle,
  NvAR_Parameter_Input(Landmarks), facial_landmarks.data(),
  sizeof(NvAR_Point2f));

//Set output memory for expression coefficients
unsigned int expressionCount;
err = NvAR_GetU32(faceExpressionHandle,
  NvAR_Parameter_Config(ExpressionCount), &expressionCount);

float expressionCoeffs = new float[expressionCount];
err = NvAR_SetF32Array(faceExpressionHandle,
  NvAR_Parameter_Output(ExpressionCoefficients), expressionCoeffs,
  expressionCount);

//Set output memory for pose rotation quaternion
NvAR_Quaternion pose = new NvAR_Quaternion();
err = NvAR_SetObject(faceExpressionHandle, NvAR_Parameter_Output(Pose),
  pose, sizeof(NvAR_Quaternion));

//Optional: If desired, set memory for bounding boxes and their confidences

err = NvAR_Run(faceExpressionHandle);
```

3.4.6.2 Alternative Usage of the Facial Expression Estimation Feature

Like the alternative usage of the Landmark detection feature, the Facial Expression Estimation feature can be used to determine the detected face bounding box, the facial keypoints, and a 3D face mesh and its rendering parameters.

When you provide an input image instead of the facial keypoints of a face, the face and the facial keypoints are automatically detected and are used to run the expression estimation. This way, if `BoundingBoxes`, `Landmarks`, or both are set as optional output properties for this feature, these properties are populated with the bounding box that contains the face and the detected facial keypoints.

`ExpressionCoefficients` and `Pose` are not optional properties for this feature. To run the feature, these properties must be set with user-provided output buffers. If this feature is also run without providing facial keypoints as an input, the path to which the `ModelDir` configuration property points must also contain the face and landmark detection TRT package files. Optionally, `CUDAStream` and the `Temporal` flag can be set for those features.

The expression coefficients can be used to drive the expressions of an avatar.

Note

The facial keypoints or the face bounding box that were determined internally can be queried from this feature but are not required for the feature to run.

The following example uses the Facial Expression Estimation feature to obtain the face expression coefficients directly from the image, without explicitly running Landmark Detection or Face Detection:

```
//Set input image buffer instead of providing facial keypoints
NvAR_SetObject(faceExpressionHandle, NvAR_Parameter_Input(Image),
  &inputImageBuffer, sizeof(NvCVImage));

//Set output memory for expression coefficients
unsigned int expressionCount;
err = NvAR_GetU32(faceExpressionHandle,
  NvAR_Parameter_Config(ExpressionCount), &expressionCount);
float expressionCoeffs = new float[expressionCount];
err = NvAR_SetF32Array(faceExpressionHandle,
  NvAR_Parameter_Output(ExpressionCoefficients), expressionCoeffs,
  expressionCount);

//Set output memory for pose rotation quaternion
NvAR_Quaternion pose = new NvAR_Quaternion();
err = NvAR_SetObject(faceExpressionHandle, NvAR_Parameter_Output(Pose),
  pose, sizeof(NvAR_Quaternion));

//Optional: Set facial keypoints as an output
NvAR_SetObject(faceExpressionHandle, NvAR_Parameter_Output(Landmarks),
  facial_landmarks.data(),sizeof(NvAR_Point2f));

//Optional: Set output memory for bounding boxes or other parameters,
//such as pose, bounding box confidence, and landmarks confidence

NvAR_Run(faceExpressionHandle);
```

3.4.6.3 Facial Expression Estimation Tracking for Temporal Frames (Videos)

If the `Temporal` flag is set and face and landmark detection are run internally, these features are optimized for temporally related frames. This means that face and facial keypoints are tracked across frames and, if requested, only one bounding box is returned as an output. If the Face Detection and Landmark Detection features are explicitly used, they need their own `Temporal` flags to be set. This flag also affects the Facial Expression Estimation feature through the `NVAR_TEMPORAL_FILTER_FACIAL_EXPRESSIONS`, `NVAR_TEMPORAL_FILTER_FACIAL_GAZE`, and `NVAR_TEMPORAL_FILTER_ENHANCE_EXPRESSIONS` bits.

3.4.7. LipSync

This section provides information about how to use the LipSync feature. LipSync uses an audio input to modify a video of a person, animating the person's lips and lower face to match the audio.

3.4.7.1 LipSync Processing

The LipSync feature takes synchronized audio samples and video frames as inputs, and produces modified video frames as output. The following example demonstrates how to process video and audio frames with the LipSync feature:

```
// Allocate source image
NvCVImage_Realloc(&source_image, source_width, source_height, NVCV_BGR, NVCV_
↪U8, NVCV_CHUNKY, NVCV_GPU, 1);

// Allocate source audio frame
std::vector<float> audio_frame(audio_frame_length);

// Allocate generated image (same resolution as the source image)
NvCVImage_Realloc(&gen_image, source_width, source_height, NVCV_BGR, NVCV_U8,
↪NVCV_CHUNKY, NVCV_GPU, 1);

// Set the video frame rate
NvAR_SetF32(lipsync_han, NvAR_Parameter_Config(VideoFPS), 30.0f);

// Optional: Set the language-specific model to load (0 = generic default, 1 =
↪German, 2 = French, 3 = Spanish).
// Must be set before NvAR_Load; requires the matching TensorRT package in
↪ModelDir.
NvAR_SetU32(lipsync_han, NvAR_Parameter_Config(Language), 0);

// Load is required before setting images
NvAR_Load(lipsync_han);

// Set input and output images
NvAR_SetObject(lipsync_han, NvAR_Parameter_Input(Image), &source_image,
↪sizeof(NvCVImage));
NvAR_SetF32Array(lipsync_han, NvAR_Parameter_Input(AudioFrameBuffer), audio_
↪frame.data(), audio_frame.size());
NvAR_SetObject(lipsync_han, NvAR_Parameter_Output(Image), &gen_image,
↪sizeof(NvCVImage));

// Run the feature
NvAR_Run(lipsync_han);
```

The `Language` configuration parameter specifies which LipSync model weights are loaded (`lipsync`, `lipsync_DE`, `lipsync_FR`, or `lipsync_ES` engines under `ModelDir`). If you do not set it, the generic (`0`) model is used. Values outside the documented range are rejected.

3.4.7.2 Region-Based LipSync

The LipSync feature supports specifying speaker regions through the `LipSyncRegionData` input property. This property specifies the area of the frame that contains the speaker's face, as well as per-region settings such as bypass and speaking flags.

Important

LipSync supports only a single region with `is_speaking` set to a nonzero value at any given time. The feature smoothly activates or deactivates LipSync processing when the `is_speaking` flag

changes, or when regions with different `tracking_id` values are marked as `is_speaking` in subsequent frames.

If multiple regions are provided with `is_speaking` set, only one is processed. If your application tracks multiple speakers, ensure that only one region is marked as `is_speaking` in each frame.

Attention

The `SpeakerData` input property is no longer supported. Use `LipSyncRegionData` instead.

The following example demonstrates how to set up region-based LipSync:

```
// Define the speaker region
NvAR_LipSyncRegion region{};
region.bbox = {x, y, width, height};  // Face bounding box
region.tracking_id = 0;
region.bypass = 0.0f;       // 0 = fully animated, 1 = no animation
region.region_type = 0;     // 0 = ROI (face detection within ROI), 1 = face
↪box
region.is_speaking = 1;     // Non-zero if this person is speaking

NvAR_LipSyncRegionData region_data{};
region_data.regions = &region;
region_data.num_regions = 1;

NvAR_SetObject(lipsync_han, NvAR_Parameter_Input(LipSyncRegionData),
  &region_data, sizeof(NvAR_LipSyncRegionData));
```

3.4.7.3 LipSync Activation Output

An output parameter provides information about the LipSync activation, including the activation strength and the face location and size:

```
NvAR_LipSyncActivation activation{};
NvAR_SetObject(lipsync_han, NvAR_Parameter_Output(Activation),
  &activation, sizeof(NvAR_LipSyncActivation));
```

After `NvAR_Run` completes, the following information is available:

- `activation.strength`: Indication of how much the face was modified (0–1).
- `activation.center_x`, `activation.center_y`: Coordinates of face center in pixels.
- `activation.size`: Face size in pixels.

3.4.7.4 Input/Output Latency

Between the input and output video frames is a fixed amount of latency; that is, at the beginning of a video the first calls to `NvAR_Run` cause the feature to ingest the input video frame but do not generate an output frame. After a fixed number of input-only calls to `NvAR_Run`, the feature generates the first output video frame. At the end of a video, the feature generates the final output frames without requiring new input video frames.

The fixed latency between input and output can be queried using the `NumInitialFrames` parameter. The following example shows how to query the latency between input and output frames.

```
// Number of initial input video frames to process before retrieving an output
 ↪frame.
uint32_t latency_frame_cnt = 0;
NvAR_GetU32(lipsync_han, NvAR_Parameter_Config(NumInitialFrames), &latency_
 ↪frame_cnt);
```

Alternatively, you can use the `Ready` output parameter to check directly whether an output frame has been generated after each call to `NvAR_Run`.

```
unsigned int output_ready = 0;
NvAR_SetU32Array(lipsync_han, NvAR_Parameter_Output(Ready), &output_ready, 1);
```

3.4.8. Active Speaker Detection

This section provides information about how to use the Active Speaker Detection feature. Active Speaker Detection identifies which person in a video is currently speaking by analyzing both video frames and synchronized audio tracks. Typical use cases include in-studio capture (e.g. sports, news, interviews) with one or more speakers talking in turn without overlapping speech.

3.4.8.1 Input Requirements

Video: Input resolution must be between 360 × 360 and 4096 × 2160 pixels (C4K; BGR, 8-bit, GPU).

Audio: For best results, prepare audio as follows:

- Use audio from the same (original, non-translated) source as the video, with **frame-accurate** synchronization from start to end. Tracks can terminate early if the speaker stops.
- Provide **one mono track per candidate speaker** when using per-speaker audio. Each track should contain only that speaker's speech, with silence when the speaker is not talking. Ensure that tracks are **clean and isolated** (no background noise or music).
- Use *NvAR_ActiveAudioIds* when external activity information, such as diarization, is available. When external activity is not available, the feature can optionally use voice-activity detection or silent-track filtering to filter the candidate tracks.
- The sample applications also support a single mixed audio track with diarization JSON. In this mode, the application creates logical audio channels from diarization speaker IDs and sets `ActiveAudioIDs` for each frame.
- Supported sample rates are 16 kHz, 44.1 kHz, and 48 kHz.

Performance: The feature has startup latency on the order of a few seconds and is designed for real-time processing of a single speaker on supported hardware. For high-FPS input, the feature internally decimates only the sync-discriminator input cadence while preserving face tracking and output at the input frame rate.

3.4.8.2 Active Speaker Detection Processing

The Active Speaker Detection feature takes video frames and multiple audio tracks as inputs, and produces tracking data with speaker identification information as output. The following example demonstrates how to process video and audio frames with the Active Speaker Detection feature:

```
// Create feature
NvAR_FeatureHandle speaker_detection_handle;
NvAR_Create(NvAR_Feature_ActiveSpeakerDetection, &speaker_detection_handle);

// Set up CUDA stream
CUstream stream;
NvAR_CudaStreamCreate(&stream);
NvAR_SetCudaStream(speaker_detection_handle, NvAR_Parameter_
↪Config(CUDAStream), stream);

// Configure the feature
NvAR_SetString(speaker_detection_handle, NvAR_Parameter_Config(ModelDir),
↪model_path);
NvAR_SetF32(speaker_detection_handle, NvAR_Parameter_Config(VideoFPS), 30.0f);
NvAR_SetU32(speaker_detection_handle, NvAR_Parameter_Config(NumAudioStreams),
↪2);
NvAR_SetU32(speaker_detection_handle, NvAR_Parameter_Config(SampleRate),
↪44100);

// Optional: derive active audio from the input tracks and smooth short voice-
↪activity changes.
uint32_t flags = NVARACTIVESPEAKERDETECTION_FLAG_DETECT_SHOT_CHANGE |
                 NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VAD |
                 NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VA_SMOOTHING_LOW;
NvAR_SetU32(speaker_detection_handle, NvAR_Parameter_Config(Flags), flags);

uint32_t max_num_output_identities;
NvAR_GetU32(speaker_detection_handle, NvAR_Parameter_
↪Config(MaxNumOutputIdentities), &max_num_output_identities);

// Load the feature
NvAR_Load(speaker_detection_handle);

// Allocate input image (BGR, U8, GPU)
NvCVImage input_image;
NvCVImage_Realloc(&input_image, width, height, NVCV_BGR, NVCV_U8, NVCV_CHUNKY,
↪ NVCV_GPU, 1);

// Set up audio frame data for multiple audio tracks
std::vector<NvAR_AudioFrame> audio_frames(num_audio_tracks);
std::vector<std::vector<float>> audio_buffers(num_audio_tracks);

for (unsigned int i = 0; i < num_audio_tracks; ++i) {
  audio_buffers[i].resize(audio_frame_length);           // See "Variable
↪Audio Frame Size" to determine audio_frame_length
  audio_frames[i].audio_data = audio_buffers[i].data();  // Audio samples for
↪a single frame
  audio_frames[i].num_samples = audio_frame_length;
  audio_frames[i].audio_id = i;
}

NvAR_AudioFrameData audio_frame_data;
audio_frame_data.audio_frames = audio_frames.data();
```

(continues on next page)

(continued from previous page)

```
audio_frame_data.num_audio_channels = num_audio_tracks;

// Set up active audio IDs
std::vector<uint32_t> active_audio_ids = {0, 1}; // Audio tracks 0 and 1 are
 ↪active
NvAR_ActiveAudioIds active_audio_ids_data;
active_audio_ids_data.active_audio_ids = active_audio_ids.data();
active_audio_ids_data.num_active_audio_ids = active_audio_ids.size();

// Set up output tracking data
std::vector<NvAR_SpeakerTrackingBBox> output_boxes(max_num_output_identities);
NvAR_ActiveSpeakerTrackingData output_tracking_data;
output_tracking_data.boxes = output_boxes.data();
output_tracking_data.max_boxes = max_num_output_identities;

// Set up control parameters
uint32_t new_shot = NVARACTIVESPEAKERDETECTION_DETECT_SHOT_CHANGE; // Auto-
 ↪detect shot changes
uint32_t flush = 0;    // Normal processing
uint32_t ready = 0;    // Output ready status

// Set inputs and outputs
NvAR_SetObject(speaker_detection_handle, NvAR_Parameter_Input(Image), &input_
 ↪image, sizeof(NvCVImage));
NvAR_SetObject(speaker_detection_handle, NvAR_Parameter_Input(AudioFrameData),
 ↪ &audio_frame_data, sizeof(NvAR_AudioFrameData));
NvAR_SetObject(speaker_detection_handle, NvAR_Parameter_Input(ActiveAudioIDs),
 ↪ &active_audio_ids_data, sizeof(NvAR_ActiveAudioIds));
NvAR_SetU32(speaker_detection_handle, NvAR_Parameter_Input(NewShot), new_
 ↪shot);
NvAR_SetU32(speaker_detection_handle, NvAR_Parameter_Input(Flush), flush);

NvAR_SetObject(speaker_detection_handle, NvAR_Parameter_
 ↪Output(ActiveSpeakerTrackingData), &output_tracking_data, sizeof(NvAR_
 ↪ActiveSpeakerTrackingData));

// Run the feature
NvAR_Run(speaker_detection_handle);
NvAR_GetU32(speaker_detection_handle, NvAR_Parameter_Output(Ready), &ready);

// Check if output is ready
if (ready != 0) {
  // Process the tracking results
  for (unsigned int i = 0; i < output_tracking_data.num_boxes; ++i) {
    const NvAR_SpeakerTrackingBBox& bbox = output_tracking_data.boxes[i];
    // bbox.bbox contains the face bounding box (x, y, width, height)
    // bbox.tracking_id contains the unique person identifier
    // bbox.audio_id contains the associated audio track ID (-1 if none)
    // bbox.confidence contains the detection confidence score
    // bbox.is_speaking indicates if this person is currently speaking
  }
}
```

3.4.8.3 Multiple Audio Tracks

The Active Speaker Detection feature supports processing multiple audio tracks simultaneously to determine which person is speaking. Each audio track is identified by a unique `audio_id` and can be selectively enabled using the `ActiveAudioIDs` input parameter. Each audio track must contain speech from one and only one speaker.

The following example shows how to configure multiple audio tracks:

```
// Set up 4 audio tracks with different audio IDs
std::vector<NvAR_AudioFrame> audio_frames(4);
std::vector<std::vector<float>> audio_buffers(4);

for (unsigned int i = 0; i < 4; ++i) {
  audio_buffers[i].resize(audio_frame_length);
  audio_frames[i].audio_data = audio_buffers[i].data();
  audio_frames[i].num_samples = audio_frame_length;
  audio_frames[i].audio_id = i; // Audio IDs: 0, 1, 2, 3
}

// Only activate audio tracks 0 and 2 for this frame
std::vector<uint32_t> active_audio_ids = {0, 2};
NvAR_ActiveAudioIds active_audio_ids_data;
active_audio_ids_data.active_audio_ids = active_audio_ids.data();
active_audio_ids_data.num_active_audio_ids = 2;
```

3.4.8.4 Voice-Activity Detection and Smoothing

Active Speaker Detection can optionally filter audio activity before speaker matching. Use this feature when the application has candidate speaker tracks but does not already have reliable per-frame activity metadata.

- `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VAD` runs denoiser voice-activity detection (VAD) on the supplied audio tracks and filters inactive tracks before audio-visual matching.
- `NVARACTIVESPEAKERDETECTION_FLAG_FILTER_SILENT_TRACKS` applies simpler silent-track filtering for tracks that are expected to be zero when inactive. It is mutually exclusive with denoiser VAD.
- `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VA_SMOOTHING_LOW` and `NVARACTIVESPEAKERDETECTION_FLAG_ENABLE_VA_SMOOTHING_HIGH` reduce short voice-activity flicker. Use only one smoothing mode at a time.

When neither denoiser VAD nor silent-track filtering is enabled, `ActiveAudioIDs` is treated as the authoritative active-audio list for each frame. When one of these filtering modes is enabled, `ActiveAudioIDs` represents the candidate audio IDs that can be filtered by the SDK.

3.4.8.5 Diarization JSON Input in Sample Applications

The SDK feature consumes audio samples and `ActiveAudioIDs`. It does not parse diarization JSON directly. The Active Speaker Detection sample applications can parse diarization JSON and use it to set `ActiveAudioIDs` for each frame.

Use this mode when you have one mixed audio track and word-level diarization metadata instead of separate per-speaker WAV files. When `--diarization` is used, provide exactly one mixed WAV file for each video stream. The sample application creates `max(speaker_id) + 1` logical audio channels

that all share the same waveform, and the diarization file determines which logical audio IDs are active for each frame.

The diarization JSON must contain a `words` array. Each word entry must include the following fields:

- `start`: Word start time, in seconds.
- `end`: Word end time, in seconds.
- `speaker_id`: Non-negative integer speaker ID. This value maps directly to the logical audio ID.

Other fields, such as word text, are ignored. Words must be ordered by start time per speaker.

Example:

```
{
  "words": [
    { "start": 0.00, "end": 0.32, "speaker_id": 0 },
    { "start": 0.45, "end": 0.80, "speaker_id": 1 }
  ]
}
```

3.4.8.6 Triton Multi-View Mode

For general information about creating Triton feature instances, connecting to an AR SDK Triton server, and configuring Triton execution, see the NVIDIA Triton Inference Server User Guide for AR and VFX SDKs.

For Triton deployments with multiple camera streams showing the same scene, the `ActiveSpeakerDetectionMultiView` model provides consistent face tracking IDs across streams. The Triton client sample enables this model with `--multiview`:

```
./ActiveSpeakerDetectionTritonClientApp --multiview --src_videos=cam0.mp4,
↪cam1.mp4 --src_audios=audio0.wav,audio1.wav
```

List all camera views from the same scene in one `--src_videos` argument, and provide one `--src_audios` entry for each video. If no audio is available for the multi-view test data, pass a silence WAV once per video.

Multi-view mode changes only the meaning of `tracking_id`: In the default model, IDs are local to each stream; in multi-view mode, IDs are assigned from a shared face gallery across streams in the Triton backend process. The output shape is unchanged.

The speaking-state fields remain stream-local. The same global `tracking_id` can have different `is_speaking`, `audio_id`, and confidence values on different streams when views have different face angle, size, occlusion, or sync evidence.

3.4.8.7 Multi-GPU Multi-View Deployment

To run multi-view Active Speaker Detection on multiple GPUs, configure the `instance_group` in `arsdk_model_repo/ActiveSpeakerDetectionMultiView/config.pbtxt` and restart the Triton server. The client command remains the same; Triton routes requests across the configured model instances.

For one instance on each of GPU 0 and GPU 1, list both GPUs in the same `instance_group` entry:

```
instance_group [
  {
```

(continues on next page)

(continued from previous page)

```
    count: 1
    kind: KIND_GPU
    gpus: [0, 1]
  }
]
```

Triton creates `count` instances per listed GPU. Use the same `--multiview` client invocation after the server restart.

3.4.8.8 Shot Change Detection

Important

`NewShot` is supported but deprecated in favor of `NvAR_Parameter_Config(Flags)`. Use with `NVARACTIVESPEAKERDETECTION_FLAG_SHOT_CHANGED` or `NVARACTIVESPEAKERDETECTION_FLAG_DETECT_SHOT_CHANGE` instead.

- `NVARACTIVESPEAKERDETECTION_DETECT_SHOT_CHANGE` (255): Automatic shot change detection.
- `NVARACTIVESPEAKERDETECTION_SHOT_UNCHANGED` (0): No shot change occurred.
- `NVARACTIVESPEAKERDETECTION_SHOT_CHANGED` (1): Manual indication of shot change.

The feature includes automatic shot change detection to maintain tracking consistency across video cuts. The `NewShot` input parameter controls this behavior:

```
// Automatic shot change detection (default)
uint32_t new_shot = NVARACTIVESPEAKERDETECTION_DETECT_SHOT_CHANGE;

// Manual shot change indication
uint32_t new_shot = NVARACTIVESPEAKERDETECTION_SHOT_CHANGED;

// Explicitly indicate no shot change
uint32_t new_shot = NVARACTIVESPEAKERDETECTION_SHOT_UNCHANGED;
```

3.4.8.9 Output Synchronization

The feature uses an internal buffer to accumulate frames before producing output. The `Ready` output parameter indicates when valid output data are available. The output is offset by the number of calls made to *NvAR_Run()* before the first output is generated.

```
uint32_t ready = 0;
NvAR_SetU32Array(speaker_detection_handle, NvAR_Parameter_Output(Ready), &
↪ready, 1);

// After NvAR_Run()
if (ready != 0) {
  // Output data in output_tracking_data is valid
  // Process the tracking results...
} else {
```

(continues on next page)

(continued from previous page)

```
  // Still accumulating frames, output not ready yet
}
```

3.4.8.10 Flush Mode

After reaching the end of input data, use flush mode to retrieve final outputs without providing new input frames:

```
// Enable flush mode at end of stream
uint32_t flush = 1;
NvAR_SetU32Array(speaker_detection_handle, NvAR_Parameter_Input(Flush), &
↪flush, 1);

// Continue calling NvAR_Run() until the return code of the feature is NVCV_ERR_
↪EOF
```

3.4.8.11 Variable Audio Frame Size

When the number of audio samples per video frame is not a whole number (for example, a sample rate of 16,000 Hz at 30 FPS yields 533.33 samples per frame), the number of audio samples should alternate to maintain proper audio-video synchronization.

The following example demonstrates how to calculate the exact number of audio samples for each frame:

```
// Configuration
const float video_fps = 30.0f;
const uint32_t sample_rate = 16000;
const double frame_duration = 1.0 / video_fps;  // 0.0333... seconds per frame

// State tracking
uint32_t last_audio_end_sample = 0;
uint32_t frame_count = 0;

// For each video frame
while (processing_video) {
  const double frame_timestamp = frame_count * frame_duration;

  // Calculate exact audio window for this frame
  const uint32_t audio_start_sample = last_audio_end_sample;
  const uint32_t audio_end_sample =
      static_cast<uint32_t>((frame_timestamp + frame_duration) * sample_rate);
  const uint32_t audio_frame_length = audio_end_sample - audio_start_sample;

  // Update tracking for next frame
  last_audio_end_sample = audio_end_sample;

  // Set the same number of samples for all tracks
  for (unsigned int track_idx = 0; track_idx < num_audio_tracks; ++track_idx)
↪{
    audio_frames[track_idx].num_samples = audio_frame_length;
  }
```

(continues on next page)

(continued from previous page)

```
  // audio_frame_length will alternate between 533 and 534 samples
  // to maintain precise audio-video synchronization

  frame_count++;
  // ... set other inputs and run NvAR_Run() ...
}
```

This approach ensures the following result:

- Frame 0: 533 samples (indices 0 to 532)
- Frame 1: 534 samples (indices 533 to 1066)
- Frame 2: 533 samples (indices 1067 to 1599)
- Frame 3: 533 samples (indices 1600 to 2132)
- And so on...

The sample range is half-open: `audio_start_sample` is inclusive and `audio_end_sample` is exclusive, so each sample belongs to exactly one frame and no samples are reused. Do not set `audio_start_sample` to `last_audio_end_sample + 1`; that would skip one sample per frame and break synchronization.

The alternating sample counts maintain audio-video synchronization by accumulating the fractional parts of the samples-per-frame calculation.

3.5. Using Multiple GPUs

Applications that are developed with the AR SDK can be used with multiple GPUs. By default, the SDK determines which GPU to use based on the capability of the currently selected GPU. If the currently selected GPU supports the SDK, the SDK uses it. Otherwise, the SDK selects the best GPU.

You can control which GPU is used in a multi-GPU environment by using the `cudaSetDevice(int whichGPU)` and `cudaGetDevice(int *whichGPU)` NVIDIA CUDA Toolkit functions and the `NvAR_SetS32(NULL, NvAR_Parameter_Config(GPU), whichGPU)` AR SDK Set function. The `Set()` function is called only once for the AR SDK before any effects are created. Because you can't transparently pass images that are allocated on one GPU to another GPU, you must ensure that the same GPU is used for all AR features.

```
NvCV_Status err;
int chosenGPU = 0; // or whatever GPU you want to use
err = NvAR_SetS32(NULL, NvAR_Parameter_Config(GPU), chosenGPU);
if (NVCV_SUCCESS != err) {
  printf("Error choosing GPU %d: %s\n", chosenGPU,
         NvCV_GetErrorStringFromCode(err));
}

cudaSetDevice(chosenGPU);
NvCVImage dst = new NvCVImage(
   // your parameters
   );
```

(continues on next page)

(continued from previous page)

```
NvAR_Handle eff;
err = NvAR_API NvAR_CreateEffect(code, &eff);

// your program logic here

err = NvAR_API NvAR_Load(eff);
err = NvAR_API NvAR_Run(eff, true);
// switch GPU for other task, then switch back for next frame
```

Buffers need to be allocated on the selected GPU, so before you allocate images on the GPU, call `cudaSetDevice()`. Neural networks need to be loaded on the selected GPU, so before `NvAR_Load()` is called, set this GPU as the current device.

To use the buffers and models, before you call `NvAR_Run()`, set the GPU device as the current device. A previous call to `NvAR_SetS32(NULL, NvAR_Parameter_Config(GPU), whichGPU)` helps enforce this requirement.

For performance reasons, the application is responsible for switching to the appropriate GPU.

3.5.1. Default Behavior in Multi-GPU Environments

The `NvAR_Load()` function internally calls `cudaGetDevice()` to identify the currently selected GPU.

The function checks the compute capability of the currently selected GPU (default 0) to determine whether the GPU architecture supports the AR SDK and completes one of the following tasks:

- If the SDK is supported, `NvAR_Load()` uses the GPU.
- If the SDK is not supported, `NvAR_Load()` searches for the most powerful GPU that supports the AR SDK and calls `cudaSetDevice()` to set that GPU as the current GPU.

If you do not require your application to use a specific GPU in a multi-GPU environment, the default behavior should suffice.

3.5.2. Selecting the GPU for AR SDK Processing in a Multi-GPU Environment

Your application might be designed to perform the task of applying an AR filter by using only a specific GPU in a multi-GPU environment. In this situation, ensure that the SDK does not override your choice of GPU for applying the video effect filter.

```
// Initialization
cudaGetDevice(&beforeGPU);

err = NvAR_Load(eff);

if (NVCV_SUCCESS != err) {
  printf("Cannot load ARSDK: %s\n", NvCV_GetErrorStringFromCode(err));
  exit(-1);
}

cudaGetDevice(&arsdkGPU);
```

(continues on next page)

(continued from previous page)

```
if (beforeGPU != arsdkGPU) {
  printf("GPU #%d cannot run AR SDK, so GPU #%d was chosen instead\n",
         beforeGPU, arsdkGPU);
}
```

3.5.3. Selecting Different GPUs for Different Tasks

Your application might be designed to perform multiple tasks in a multi-GPU environment, such as rendering a game and applying an AR filter. In this situation, select the best GPU for each task before calling `NvAR_Load()`.

1. Call `cudaGetDeviceCount()` to determine the number of GPUs in your environment.

   ```
   // Get the number of GPUs
   cuErr = cudaGetDeviceCount(&deviceCount);
   ```

2. Get the properties of each GPU and determine whether it is the best GPU for each task by performing the following operations for each GPU in a loop:

   a. Call `cudaSetDevice()` to set the current GPU.

   b. Call `cudaGetDeviceProperties()` to get the properties of the current GPU.

   c. To determine whether the GPU is the best GPU for each specific task, use a custom code in your application to analyze the properties that were retrieved by `cudaGetDeviceProperties()`.

   The following example uses the compute capability to determine whether a GPU's properties should be analyzed and whether the current GPU is the best GPU on which to apply a video effect filter. A GPU's properties are analyzed only when the compute capability is 7.5, 8.6, or 8.9, which denotes a GPU that is based on Turing, the Ampere architecture, or the Ada architecture, respectively.

   ```
   // Loop through the GPUs to get the properties of each GPU and
   // determine whether it is the best GPU for each task based on the
   // properties obtained.
   for (int dev = 0; dev < deviceCount; ++dev) {
      cudaSetDevice(dev);
      cudaGetDeviceProperties(&deviceProp, dev);

      if (DeviceIsBestForARSDK(&deviceProp)) gpuARSDK = dev;
      if (DeviceIsBestForGame(&deviceProp)) gpuGame = dev;
      // your program logic here
   }

   cudaSetDevice(gpuARSDK);
   err = NvAR_Set...; // set parameters
   err = NvAR_Load(eff);
   ```

3. In the loop to complete the application's tasks, select the best GPU for each task before performing the task.

   a. Call `cudaSetDevice()` to select the GPU for the task.

b. Make all the function calls required to perform the task.

In this way, you select the best GPU for each task only once without setting the GPU for every function call.

This example selects the best GPU for rendering a game and uses custom code to render the game. It then selects the best GPU for applying a video effect filter before calling the `NvCVImage_Transfer()` and `NvAR_Run()` functions to apply the filter, avoiding the need to save and restore the GPU for every NVIDIA AR SDK API call.

```
// Select the best GPU for each task and perform the task.
while (!done) {
   // your program logic here
   cudaSetDevice(gpuGame);
   RenderGame();

   cudaSetDevice(gpuARSDK);
   err = NvAR_Run(eff, 1);
   // your program logic here
}
```

3.5.4. Using Multi-Instance GPU (Linux only)

Applications that are developed with the AR SDK can be deployed on Multi-Instance GPU (MIG) on supported devices, such as NVIDIA DGX™ A100. MIG lets you partition a device into up to seven multiple GPU instances, each with separate streaming multiprocessors, separate slices of the GPU memory, and separate pathways to the memory. This process ensures that heavy resource usage by an application on one partition does not impact the performance of the applications running on other partitions.

To run an application on a MIG partition, you do not need to call any additional SDK API in your application. You can specify which MIG instance to use for execution during invocation of your application.

You can select the MIG instance using one of the following options:

- The bare-metal method of using the `CUDA_VISIBLE_DEVICES` environment variable.
- The container method by using the NVIDIA Container Toolkit. MIG is supported only on Linux.

Refer to the NVIDIA Multi-Instance GPU User Guide for more information about the MIG and its usage.

Chapter 4. AR SDK API Reference

This section provides detailed information about the APIs in the NVIDIA® AR SDK.

4.1. Structures

This section provides information about the structures in the AR SDK APIs. The structures are defined in the following header files:

- `nvAR.h`
- `nvAR_defs.h` (mostly data types)

4.1.1. NvAR_BBoxes

```
struct NvAR_BBoxes {
    NvAR_Rect *boxes;
    uint8_t num_boxes;
    uint8_t max_boxes;
};
```

4.1.1.1 Members

boxes

Type: NvAR_Rect*
Pointer to an array of bounding boxes that are allocated by the user.

num_boxes

Type: uint8_t
The number of bounding boxes in the array.

max_boxes

Type: uint8_t
The maximum number of bounding boxes that can be stored in the array as defined by the user.

4.1.1.2 Remarks

This structure is returned as the output of the face detection feature.

Defined in: nvAR_defs.h

4.1.2. NvAR_TrackingBBox

```
struct NvAR_TrackingBBox {
    NvAR_Rect bbox;
    uint16_t tracking_id;
};
```

4.1.2.1 Members

bbox

Type: NvAR_Rect
Bounding box that is allocated by the user.

tracking_id

Type: uint16_t
The Tracking ID assigned to the bounding box by Multi-Person Tracking.

4.1.2.2 Remarks

This structure is returned as the output of the body pose feature when multi-person tracking is enabled.

Defined in: nvAR_defs.h

4.1.3. NvAR_TrackingBBoxes

```
struct NvAR_TrackingBBoxes {
    NvAR_TrackingBBox *boxes;
    uint8_t num_boxes;
    uint8_t max_boxes;
};
```

4.1.3.1 Members

boxes

Type: NvAR_TrackingBBox *
Pointer to an array of tracking bounding boxes that are allocated by the user.

num_boxes

Type: uint8_t

The number of bounding boxes in the array.

max_boxes

Type: uint8_t
The maximum number of bounding boxes that can be stored in the array as defined by the user.

4.1.3.2 Remarks

This structure is returned as the output of the body pose feature when multi-person tracking is enabled.

Defined in: nvAR_defs.h

4.1.4. NvAR_FaceMesh

```
struct NvAR_FaceMesh {
    NvAR_Vec3<float> *vertices;
    size_t num_vertices;
    NvAR_Vec3<unsigned short> *tvi;
    size_t num_triangles;
};
```

4.1.4.1 Members

vertices

Type: NvAR_Vec3<float>*
Pointer to an array of vectors that represent the mesh 3D vertex positions.

num_vertices

Type: size_t
The number of vertices in the array pointed to by the vertices parameter.

tvi

Type: NvAR_Vec3<unsigned short> *
Pointer to an array of vectors that represent the mesh triangle's vertex indices.

num_triangles

Type: size_t
The number of mesh triangles.

4.1.4.2 Remarks

This structure is returned as an output of the Mesh Tracking feature.

Defined in: nvAR_defs.h

4.1.5. NvAR_Frustum

```
struct NvAR_Frustum {
    float left = -1.0f;
    float right = 1.0f;
    float bottom = -1.0f;
    float top = 1.0f;
};
```

4.1.5.1 Members

left

Type: float
The X coordinate of the top-left corner of the viewing frustum.

right

Type: float
The X coordinate of the bottom-right corner of the viewing frustum.

bottom

Type: float
The Y coordinate of the bottom-right corner of the viewing frustum.

top

Type: float
The Y coordinate of the top-left corner of the viewing frustum.

4.1.5.2 Remarks

This structure represents a camera viewing frustum for an orthographic camera. As a result, it contains only the left, the right, the top, and the bottom coordinates in pixels. It does **not** contain a near or a far clipping plane.

Defined in: nvAR_defs.h

4.1.6. NvAR_FeatureHandle

```
typedef struct nvAR_Feature *NvAR_FeatureHandle;
```

4.1.6.1 Remarks

This type defines the handle of a feature that is defined by the SDK. It is used to reference the feature at runtime when the feature is executed and must be destroyed when it is no longer required.

Defined in: nvAR_defs.h

4.1.7. NvAR_Point2f

```
typedef struct NvAR_Point2f {
    float x, y;
} NvAR_Point2f;
```

4.1.7.1 Members

x

Type: float
The X coordinate of the point in pixels.

y

Type: float
The Y coordinate of the point in pixels.

4.1.7.2 Remarks

This structure represents the X and Y coordinates of one point in 2D space.

Defined in: nvAR_defs.h

4.1.8. NvAR_Point3f

```
typedef struct NvAR_Point3f {
    float x, y, z;
} NvAR_Point3f;
```

4.1.8.1 Members

x

Type: float
The X coordinate of the point in pixels.

y

Type: float
The Y coordinate of the point in pixels.

z

Type: float
The Z coordinate of the point in pixels.

4.1.8.2 Remarks

This structure represents the X, Y, Z coordinates of one point in 3D space.

Defined in: nvAR_defs.h

4.1.9. NvAR_Quaternion

```
struct NvAR_Quaternion {
    float x, y, z, w;
};
```

4.1.9.1 Members

x

Type: float
The first coefficient of the complex part of the quaternion.

y

Type: float
The second coefficient of the complex part of the quaternion.

z

Type: float
The third coefficient of the complex part of the quaternion.

w

Type: float
The scalar coefficient of the quaternion.

4.1.9.2 Remarks

This structure represents the coefficients in the quaternion that are expressed in the following equation: q = w + xi + yj + zk

Defined in: nvAR_defs.h

4.1.10. NvAR_Transform3f

```
typedef struct NvAR_Transform3f {
    NvAR_Quaternion rotation;
    NvAR_Point3f translation;
} NvAR_Transform3f;
```

4.1.10.1 Members

rotation

Type: NvAR_Quaternion
Root rotation as an XYZW quaternion.

translation

Type: NvAR_Point3f
Root translation in camera coordinates, in meters.

4.1.10.2 Remarks

The 3D Body Pose `RootPose` output uses this structure.

Defined in: nvAR3DBodyPoseEstimation.h

4.1.11. NvAR_Rect

```
typedef struct NvAR_Rect {
    float x, y, width, height;
} NvAR_Rect;
```

4.1.11.1 Members

x

Type: float
The X coordinate of the top left corner of the bounding box in pixels.

y

Type: float
The Y coordinate of the top left corner of the bounding box in pixels.

width

Type: float
The width of the bounding box in pixels.

height

Type: float
The height of the bounding box in pixels.

4.1.11.2 Remarks

This structure represents the position and size of a rectangular 2D bounding box.

Defined in: nvAR_defs.h

4.1.12. NvAR_RenderingParams

```
struct NvAR_RenderingParams {
    NvAR_Frustum frustum;
    NvAR_Quaternion rotation;
    NvAR_Vec3<float> translation;
};
```

4.1.12.1 Members

frustum

Type: NvAR_Frustum
The camera viewing frustum for an orthographic camera.

rotation

Type: NvAR_Quaternion
The rotation of the camera relative to the mesh.

translation

Type: NvAR_Vec3<float>
The translation of the camera relative to the mesh.

4.1.12.2 Remarks

This structure defines the parameters that are used to draw a 3D face mesh in a window on the computer screen so that the face mesh is aligned with the corresponding video frame. The projection matrix is constructed from the frustum parameter, and the model view matrix is constructed from the rotation and translation parameters.

Defined in: nvAR_defs.h

4.1.13. NvAR_Vector2f

```
typedef struct NvAR_Vector2f {
    float x, y;
} NvAR_Vector2f;
```

4.1.13.1 Members

x

Type: float
The X component of the 2D vector.

y

Type: float
The Y component of the 2D vector.

4.1.13.2 Remarks

This structure represents a 2D vector.

Defined in: nvAR_defs.h

4.1.14. NvAR_Vector3f

```
typedef struct NvAR_Vector3f {
    float vec[3];
} NvAR_Vector3f;
```

4.1.14.1 Members

vec

Type: float array of size 3
A vector of size 3.

4.1.14.2 Remarks

This structure represents a 3D vector.

Defined in: nvAR_defs.h

4.1.15. NvAR_Vector3u16

```
typedef struct NvAR_Vector3u16 {
    unsigned short vec[3];
} NvAR_Vector3u16;
```

4.1.15.1 Members

vec

Type: unsigned short array of size 3
A vector of size 3.

4.1.15.2 Remarks

This structure represents a 3D vector.

Defined in: nvAR_defs.h

4.1.16. NvAR_SpeakerData (Deprecated)

Deprecated since version 1.1.0.1: `NvAR_SpeakerData` is no longer supported. Use *NvAR_LipSyncRegionData* instead.

```
typedef struct NvAR_SpeakerData {
    const float* audio_frame_data;
    size_t audio_frame_size;
    NvAR_Rect region;
    uint8_t region_type;
    float bypass;
} NvAR_SpeakerData;
```

4.1.16.1 Members

audio_frame_data

Type: const float*
Pointer to a buffer containing driving audio. The audio is assumed to be mono and in float format.

audio_frame_size

Type: size_t
Number of audio samples in the audio frame.

region

Type: NvAR_Rect
Region that contains the speaker's face.

region_type

Type: uint8_t
Flag that indicates the type of region. 0 = ROI, the feature should perform face detection within the ROI. 1 = face box, the feature should skip face detection.

bypass

Type: float
Value in the range 0–1 that can suppress speaker animation by reducing the opacity of the animated face. When the value is 0, the speaker's face is animated according to the feature's internal logic. As the value increases, the opacity of the animated face is reduced, blending with the original image. When the value is 1, the face is not animated.

Defined in: nvAR_defs.h

4.1.16.2 Remarks

This structure represents the input data that is used to animate a speaker's face in a video frame.

4.1.17. NvAR_LipSyncRegion

```
typedef struct NvAR_LipSyncRegion {
    NvAR_Rect bbox;
    uint16_t tracking_id;
    int16_t audio_id;
    float bypass;
    uint8_t region_type;
    uint8_t is_speaking;
} NvAR_LipSyncRegion;
```

4.1.17.1 Members

bbox

Type: NvAR_Rect
Bounding box that contains the speaker's face.

tracking_id

Type: uint16_t
Tracking ID that uniquely identifies the speaker across frames.

audio_id

Type: int16_t
Reserved for future use.

bypass

Type: float
Value in the range 0–1 that can reduce the output opacity. When the value is 0, the speaker's face is animated according to the feature's internal logic. As the value increases, the opacity of the animated face is reduced, blending with the original image. When the value is 1, the face is not animated.

region_type

Type: uint8_t
Flag that indicates the type of region. 0 = ROI, the feature should perform face detection within the ROI. 1 = face box, the feature should skip face detection.

is_speaking

Type: uint8_t
Flag that indicates whether the speaker is currently speaking. When set to a non-zero value, the region is considered to be speaking and LipSync animation is applied.

4.1.17.2 Remarks

This structure represents a single speaker region used as input to the LipSync feature. It replaces the deprecated `NvAR_SpeakerData` structure with additional multi-speaker support through tracking IDs and a speaking flag.

Defined in: nvARLipSync.h

4.1.18. NvAR_LipSyncRegionData

```
typedef struct NvAR_LipSyncRegionData {
    NvAR_LipSyncRegion* regions;
    uint8_t num_regions;
} NvAR_LipSyncRegionData;
```

4.1.18.1 Members

regions

Type: NvAR_LipSyncRegion*
Pointer to an array of *NvAR_LipSyncRegion* structures that define the speaker regions in the frame.

num_regions

Type: uint8_t
The number of regions in the array.

4.1.18.2 Remarks

This structure specifies the regions and per-region settings for the LipSync feature. It replaces the deprecated `NvAR_SpeakerData` input. When this input is not set (nullptr), the LipSync feature operates in single full-frame region mode.

Defined in: nvARLipSync.h

4.1.19. NvAR_LipSyncActivation

```
typedef struct NvAR_LipSyncActivation {
    float strength;
    float center_x;
    float center_y;
    float size;
} NvAR_LipSyncActivation;
```

4.1.19.1 Members

strength

Type: float

Value in the range 0–1 representing the LipSync activation strength. When the value is 0, the original face was copied directly to the output without modification. When the value is 1, the original face was completely replaced by the animated face in the output.

center_x

Type: float
The X coordinate of the center of the face in pixels.

center_y

Type: float
The Y coordinate of the center of the face in pixels.

size

Type: float
The face size in pixels.

4.1.19.2 Remarks

This structure is returned as an output of the LipSync feature and provides detailed information about the activation strength and the location and size of the face that was animated.

Defined in: nvARLipSync.h

4.1.20. NvAR_AudioFrame

```
typedef struct NvAR_AudioFrame {
    const float* audio_data;
    uint32_t num_samples;
    int32_t audio_id;
} NvAR_AudioFrame;
```

4.1.20.1 Members

audio_data

Type: const float*
Pointer to audio samples in floating-point format, ranging from -1.0 to 1.0.

num_samples

Type: uint32_t
Number of audio samples in this frame.

audio_id

Type: int32_t
Unique identifier for this audio track.

Defined in: nvARActiveSpeakerDetection.h

4.1.20.2 Remarks

This structure represents a single audio frame from one audio track for Active Speaker Detection.

4.1.21. NvAR_AudioFrameData

```
typedef struct NvAR_AudioFrameData {
    const NvAR_AudioFrame* audio_frames;
    uint32_t num_audio_channels;
} NvAR_AudioFrameData;
```

4.1.21.1 Members

audio_frames

Type: const NvAR_AudioFrame*
Array of audio frames, one per audio channel or track.

num_audio_channels

Type: uint32_t
Number of audio channels or tracks in the `audio_frames` array.

Defined in: nvARActiveSpeakerDetection.h

4.1.21.2 Remarks

This structure contains audio frame data for all audio tracks to be processed by Active Speaker Detection.

4.1.22. NvAR_ActiveAudioIds

```
typedef struct NvAR_ActiveAudioIds {
    const uint32_t* active_audio_ids;
    uint32_t num_active_audio_ids;
} NvAR_ActiveAudioIds;
```

4.1.22.1 Members

active_audio_ids

Type: const uint32_t*
Array of audio IDs that are currently active and should be processed.

num_active_audio_ids

Type: uint32_t
Number of active audio IDs in the `active_audio_ids` array.

Defined in: nvARActiveSpeakerDetection.h

4.1.22.2 Remarks

This structure specifies which audio tracks are active and should be considered for speaker detection. Applications can populate this list from their own activity logic or from diarization metadata. The Active Speaker Detection sample applications can parse diarization JSON and set this structure for each frame.

4.1.23. NvAR_SpeakerTrackingBBox

```
typedef struct NvAR_SpeakerTrackingBBox {
    NvAR_Rect bbox;
    uint16_t tracking_id;
    int16_t audio_id;
    float confidence;
    uint8_t is_speaking;
} NvAR_SpeakerTrackingBBox;
```

4.1.23.1 Members

bbox

Type: NvAR_Rect
Bounding box of the detected face (x, y, width, height).

tracking_id

Type: uint16_t
Unique identifier for a person tracked across frames.

audio_id

Type: int16_t
ID of the audio track associated with this speaker, or -1 if no audio is associated.

confidence

Type: float
Detection confidence score for this face.

is_speaking

Type: uint8_t
Flag indicating whether this person is currently speaking (1) or not (0).

Defined in: nvARActiveSpeakerDetection.h

4.1.23.2 Remarks

This structure represents a single tracked face with speaker detection information.

4.1.24. NvAR_ActiveSpeakerTrackingData

```
typedef struct NvAR_ActiveSpeakerTrackingData {
    NvAR_SpeakerTrackingBBox* boxes;
    uint8_t num_boxes;
    uint8_t max_boxes;
} NvAR_ActiveSpeakerTrackingData;
```

4.1.24.1 Members

boxes

Type: NvAR_SpeakerTrackingBBox*
Array of tracked faces with speaker detection information.

num_boxes

Type: uint8_t
Number of valid entries in the `boxes` array.

max_boxes

Type: uint8_t
Maximum number of boxes that can be stored in the array.

Defined in: nvARActiveSpeakerDetection.h

4.1.24.2 Remarks

The bounding box array must be pre-allocated by the user. The user owns the memory pointed to by `boxes`. The value of ``max_boxes` must correspond to the size of the array; `num_boxes` is set by the feature.

This structure contains the output data from Active Speaker Detection, including all tracked faces and their speaking status.

4.2. Functions

This section provides information about the functions in the AR SDK.

4.2.1. NvAR_Create

```
NvAR_Result NvAR_Create(
    NvAR_FeatureID featureID,
    NvAR_FeatureHandle *handle
);
```

4.2.1.1 Parameters

featureID [in]

Type: NvAR_FeatureID
The type of feature to be created.

handle [out]

Type: NvAR_FeatureHandle*
A handle to the newly created feature instance.

4.2.1.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_FEATURENOTFOUND
- NVCV_ERR_INITIALIZATION

4.2.1.3 Remarks

This function creates an instance of the specified feature type and writes a handle to the feature instance to the handle out parameter.

4.2.2. NvAR_Destroy

```
NvAR_Result NvAR_Destroy(
    NvAR_FeatureHandle handle
);
```

4.2.2.1 Parameters

handle [in]

Type: NvAR_FeatureHandle
The handle to the feature instance to be released.

4.2.2.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_FEATURENOTFOUND

4.2.2.3 Remarks

This function releases the feature instance with the specified handle. Because handles are not reference counted, the handle is invalid after this function is called.

4.2.3. NvAR_Load

```
NvAR_Result NvAR_Load(
    NvAR_FeatureHandle handle
);
```

4.2.3.1 Parameters

handle [in]

Type: NvAR_FeatureHandle
The handle to the feature instance to load.

4.2.3.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_FEATURENOTFOUND
- NVCV_ERR_INITIALIZATION
- NVCV_ERR_UNIMPLEMENTED

4.2.3.3 Remarks

This function loads the specified feature instance and validates any configuration properties that were set for the feature instance.

4.2.4. NvAR_Run

```
NvAR_Result NvAR_Run(
    NvAR_FeatureHandle handle
);
```

4.2.4.1 Parameters

handle [in]

Type: NvAR_FeatureHandle
The handle to the feature instance to be run.

4.2.4.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_GENERAL
- NVCV_ERR_FEATURENOTFOUND

- NVCV_ERR_MEMORY
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_PARAMETER

4.2.4.3 Remarks

This function validates the input/output properties that are set by the user, runs the specified feature instance with the input properties that were set for the instance, and writes the results to the output properties set for the instance. The input and output properties are set by the accessor functions. Refer to *Summary of AR SDK Accessor Functions* for more information.

4.2.5. NvAR_GetCudaStream

```
NvAR_Result NvAR_GetCudaStream(
    NvAR_FeatureHandle handle,
    const char *name,
    const CUStream *stream
);
```

4.2.5.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the CUDA stream.

name

Type: const char*
The NvAR_Parameter_Config(CUDAStream) key value. Any other key value returns an error.

stream

Type: const CUStream*
Pointer to the CUDA stream where the CUDA stream retrieved is to be written.

4.2.5.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.5.3 Remarks

This function gets the CUDA stream in which the specified feature instance will run and writes the CUDA stream to be retrieved to the location that is specified by the stream parameter.

4.2.6. NvAR_CudaStreamCreate

```
NvCV_Status NvAR_CudaStreamCreate(
    CUstream *stream
);
```

4.2.6.1 Parameters

stream [out]

Type: CUstream *
The location in which to store the newly allocated CUDA stream.

4.2.6.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_CUDA_VALUE if a CUDA parameter is not within its acceptable range.

4.2.6.3 Remarks

This function creates a CUDA stream. It is a wrapper for the CUDA Runtime API function cudaStream-Create() that you can use to avoid linking with the NVIDIA CUDA Toolkit libraries. This function and cudaStreamCreate() are equivalent and interchangeable.

4.2.7. NvAR_CudaStreamDestroy

```
void NvAR_CudaStreamDestroy(
    CUstream stream
);
```

4.2.7.1 Parameters

stream [in]

Type: CUstream
The CUDA stream to destroy.

4.2.7.2 Return Value

Does not return a value.

4.2.7.2.1 Remarks

This function destroys a CUDA stream. It is a wrapper for the CUDA Runtime API function cudaStreamDestroy() that you can use to avoid linking with the NVIDIA CUDA Toolkit libraries. This function and cudaStreamDestroy() are equivalent and interchangeable.

4.2.8. NvAR_GetF32

```
NvAR_Result NvAR_GetF32(
    NvAR_FeatureHandle handle,
    const char *name,
    float *val
);
```

4.2.8.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the specified 32-bit floating-point parameter.

name

Type: const char*
The key value that is used to access the 32-bit float parameters.

val

Type: float*
Pointer to the 32-bit floating-point number where the value retrieved is to be written.

4.2.8.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.8.3 Remarks

This function gets the value of the specified single-precision (32-bit) floating-point parameter for the specified feature instance and writes the value to be retrieved to the location that is specified by the val parameter.

4.2.9. NvAR_GetF64

```
NvAR_Result NvAR_GetF64(
    NvAR_FeatureHandle handle,
    const char *name,
    double *val
);
```

4.2.9.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the specified 64-bit floating-point parameter.

name

Type: const char*
The key value used to access the 64-bit double parameters.

val

Type: double*
Pointer to the 64-bit double-precision floating-point number where the retrieved value will be written.

4.2.9.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.9.3 Remarks

This function gets the value of the specified double-precision (64-bit) floating-point parameter for the specified feature instance and writes the retrieved value to the location that is specified by the val parameter.

4.2.10. NvAR_GetF32Array

```
NvAR_Result NvAR_GetF32Array(
    NvAR_FeatureHandle handle,
    const char *name,
```

(continues on next page)

(continued from previous page)

```
    const float **vals,
    int *count
);
```

4.2.10.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the specified float array.

name

Type: const char*
Refer to Key Values in the Properties of a Feature Type for a complete list of key values.

vals

Type: const float**
Address of a floating-point pointer, where the retrieved pointer will be written.

count

Type: int*
Currently unused. The number of elements in the array specified by the vals parameter.

4.2.10.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.11. NvAR_GetU32Array

```
NvAR_Result NvAR_GetU32Array(
    NvAR_FeatureHandle handle,
    const char *name,
    const unsigned int **vals,
    int *count
);
```

4.2.11.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the specified unsigned integer array.

name

Type: const char*
Refer to Key Values in the Properties of a Feature Type for a complete list of key values.

vals

Type: const unsigned int**
Address of an unsigned integer pointer, where the retrieved pointer will be written.

count

Type: int*
Currently unused. The number of elements in the array specified by the vals parameter.

4.2.11.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.12. NvAR_GetObject

```
NvAR_Result NvAR_GetObject(
    NvAR_FeatureHandle handle,
    const char *name,
    const void **ptr,
    unsigned long typeSize
);
```

4.2.12.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you can get the specified object.

name

Type: const char*
Refer to Key Values in the Properties of a Feature Type for a complete list of key values.

ptr

Type: const void**
A pointer to the memory that is allocated for the objects defined in Structures.

typeSize

Type: unsigned long
The size of the item to which the pointer points. If the size does not match, an NVCV_ERR_MISMATCH is returned.

4.2.12.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.12.2.1 Remarks

This function gets the specified object for the specified feature instance and stores the object in the memory location that is specified by the ptr parameter.

4.2.13. NvAR_GetS32

```
NvAR_Result NvAR_GetS32(
    NvAR_FeatureHandle handle,
    const char *name,
    int *val
);
```

4.2.13.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you get the specified 32-bit signed integer parameter.

name

Type: const char*
The key value that is used to access the signed integer parameters.

val

Type: int*
Pointer to the 32-bit signed integer where the retrieved value will be written.

4.2.13.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.13.3 Remarks

This function gets the value of the specified 32-bit signed integer parameter for the specified feature instance and writes the retrieved value to the location that is specified by the val parameter.

4.2.14. NvAR_GetString

```
NvAR_Result NvAR_GetString(
    NvAR_FeatureHandle handle,
    const char *name,
    const char** str
);
```

4.2.14.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you get the specified character string parameter.

name

Type: const char*
Refer to Key Values in the Properties of a Feature Type for a complete list of key values.

str

Type: const char**
The address where the requested character string pointer is stored.

4.2.14.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.14.2.1 Remarks

This function gets the value of the specified character string parameter for the specified feature instance and writes the retrieved string to the location that is specified by the str parameter.

4.2.15. NvAR_GetU32

```
NvAR_Result NvAR_GetU32(
    NvAR_FeatureHandle handle,
    const char *name,
    unsigned int* val
);
```

4.2.15.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance from which you want to get the specified 32-bit unsigned integer parameter.

name

Type: const char *
The key value that is used to access the unsigned integer parameters as defined in nvAR_defs.h.

val

Type: unsigned int*
Pointer to the 32-bit unsigned integer where the retrieved value will be written.

4.2.15.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR

- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.15.2.1 Remarks

This function gets the value of the specified 32-bit unsigned integer parameter for the specified feature instance and writes the retrieved value to the location that is specified by the val parameter.

4.2.16. NvAR_GetU64

```
NvAR_Result NvAR_GetU64(
    NvAR_FeatureHandle handle,
    const char *name,
    unsigned long long *val
);
```

4.2.16.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the returned feature instance from which you get the specified 64-bit unsigned integer parameter.

name

Type: const char *
The key value used to access the unsigned 64-bit integer parameters as defined in nvAR_defs.h.

val

Type: unsigned long long*
Pointer to the 64-bit unsigned integer where the retrieved value will be written.

4.2.16.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.16.3 Remarks

This function gets the value of the specified 64-bit unsigned integer parameter for the specified feature instance and writes the retrieved value to the location specified by the val parameter.

4.2.17. NvAR_SetCudaStream

```
NvAR_Result NvAR_SetCudaStream(
    NvAR_FeatureHandle handle,
    const char *name,
    CUStream stream
);
```

4.2.17.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance that is returned for which you want to set the CUDA stream.

name

Type: const char *
The NvAR_Parameter_Config(CUDAStream) key value. Any other key value returns an error.

stream

Type: CUStream
The CUDA stream in which to run the feature instance on the GPU.

4.2.17.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.17.3 Remarks

This function sets the CUDA stream, in which the specified feature instance will run, to the parameter stream.

Defined in: nvAR.h

4.2.18. NvAR_SetF32

```
NvAR_Result NvAR_SetF32(
    NvAR_FeatureHandle handle,
    const char *name,
    float val
);
```

4.2.18.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified 32-bit floating-point parameter.

name

Type: const char *
The key value used to access the 32-bit float parameters as defined in nvAR_defs.h.

val

Type: float
The 32-bit floating-point number to which the parameter is to be set.

4.2.18.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.18.3 Remarks

This function sets the specified single-precision (32-bit) floating-point parameter for the specified feature instance to the val parameter.

4.2.19. NvAR_SetF64

```
NvAR_Result NvAR_SetF64(
    NvAR_FeatureHandle handle,
    const char *name,
    double val
);
```

4.2.19.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified 64-bit floating-point parameter.

name

Type: const char *
The key value used to access the 64-bit float parameters as defined in nvAR_defs.h.

val

Type: double
The 64-bit double-precision floating-point number to which the parameter will be set.

4.2.19.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.19.3 Remarks

This function sets the specified double-precision (64-bit) floating-point parameter for the specified feature instance to the val parameter.

4.2.20. NvAR_SetF32Array

```
NvAR_Result NvAR_SetF32Array(
    NvAR_FeatureHandle handle,
    const char *name,
    float* vals,
    int count
);
```

4.2.20.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified float array.

name

Type: const char *
The key value used to access the float array parameters as defined in nvAR_defs.h.

vals

Type: float*
An array of floating-point numbers to which the parameter will be set.

count

Type: int
Currently unused. The number of elements in the array that is specified by the vals parameter.

4.2.20.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.20.3 Remarks

This function assigns the array of floating-point numbers that are defined by the vals parameter to the specified floating-point-array parameter for the specified feature instance.

4.2.21. NvAR_SetU32Array

```
NvAR_Result NvAR_SetU32Array(
    NvAR_FeatureHandle handle,
    const char *name,
    unsigned int* vals,
    int count
);
```

4.2.21.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified unsigned int array.

name

Type: const char *

The key value used to access the unsigned integer array parameters as defined in nvAR_defs.h.

vals

Type: unsigned int*
An array of unsigned integers to which the parameter will be set.

count

Type: int
Currently unused. The number of elements in the array that is specified by the vals parameter.

4.2.21.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.21.3 Remarks

This function assigns the array of unsigned integers that are defined by the vals parameter to the specified unsigned integer-array parameter for the specified feature instance.

4.2.22. NvAR_SetObject

```
NvAR_Result NvAR_SetObject(
    NvAR_FeatureHandle handle,
    const char *name,
    void *ptr,
    unsigned long typeSize
);
```

4.2.22.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified object.

name

Type: const char *
The key value used to access the object parameters as defined in nvAR_defs.h.

ptr

Type: void *
A pointer to memory that was allocated to the objects that were defined in Structures.

typeSize

Type: unsigned long
The size of the item to which the pointer points. If the size does not match, an NVCV_ERR_MISMATCH is returned.

4.2.22.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.22.3 Remarks

This function assigns the memory of the object that was specified by the ptr parameter to the specified object parameter for the specified feature instance.

4.2.23. NvAR_SetS32

```
NvAR_Result NvAR_SetS32(
    NvAR_FeatureHandle handle,
    const char *name,
    int val
);
```

4.2.23.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified 32-bit signed integer parameter.

name

Type: const char *
The key value used to access the signed 32-bit integer parameters as defined in nvAR_defs.h.

val

Type: int
The 32-bit signed integer to which the parameter will be set.

4.2.23.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.23.3 Remarks

This function sets the specified 32-bit signed integer parameter for the specified feature instance to the val parameter.

4.2.24. NvAR_SetString

```
NvAR_Result NvAR_SetString(
    NvAR_FeatureHandle handle,
    const char *name,
    const char* str
);
```

4.2.24.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified character string parameter.

name

Type: const char *
The key value used to access the string parameters as defined in nvAR_defs.h.

str

Type: const char*
Pointer to the character string to which you want to set the parameter.

4.2.24.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL

- NVCV_ERR_MISMATCH

4.2.24.3 Remarks

This function sets the value of the specified character string parameter for the specified feature instance to the str parameter.

4.2.25. NvAR_SetU32

```
NvAR_Result NvAR_SetU32(
    NvAR_FeatureHandle handle,
    const char *name,
    unsigned int val
);
```

4.2.25.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified 32-bit unsigned integer parameter.

name

Type: const char *
The key value used to access the unsigned 32-bit integer parameters as defined in nvAR_defs.h.

val

Type: unsigned int
The 32-bit unsigned integer to which you want to set the parameter.

4.2.25.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.25.3 Remarks

This function sets the value of the specified 32-bit unsigned integer parameter for the specified feature instance to the val parameter.

4.2.26. NvAR_SetU64

```
NvAR_Result NvAR_SetU64(
    NvAR_FeatureHandle handle,
    const char *name,
    unsigned long long val
);
```

4.2.26.1 Parameters

handle

Type: NvAR_FeatureHandle
The handle to the feature instance for which you want to set the specified 64-bit unsigned integer parameter.

name

Type: const char *
The key value used to access the unsigned 64-bit integer parameters as defined in nvAR_defs.h.

val

Type: unsigned long long
The 64-bit unsigned integer to which you want to set the parameter.

4.2.26.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_PARAMETER
- NVCV_ERR_SELECTOR
- NVCV_ERR_GENERAL
- NVCV_ERR_MISMATCH

4.2.26.3 Remarks

This function sets the value of the specified 64-bit unsigned integer parameter for the specified feature instance to the val parameter.

4.2.27. NvAR_ConfigureLogger

```
NvCV_Status NvAR_ConfigureLogger(
    int verbosity,
    const char *file,
    void (*cb)(void*, const char*),
    void *cb_data
);
```

4.2.27.1 Parameters

verbosity

Type: int
The maximum level of verbosity to be recorded into the log:

* NVCV_LOG_FATAL (0) - Message printed before aborting due to unrecoverable error
* NVCV_LOG_ERROR (1) - Operation failed, but not fatal
* NVCV_LOG_WARNING (2) - Issue fixed with potential performance impact
* NVCV_LOG_INFO (3) - Informational message

Higher level verbosity includes logs from lower levels.

file

Type: const char*
Used only for stderr/file logging, otherwise NULL. Examples:

* “stderr” - Log to stderr
* “/path/to/log.txt” - Log to specified file

cb

Type: void (*)(void * data, const char * msg)
Callback function pointer with arguments:

* data - Pointer to callback data
* msg - Message to log

cb_data

Type: void*
Pointer provided as first argument to callback function

4.2.27.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_FILE if specified log file cannot be opened

4.2.27.3 Logger Types

The SDK provides several logger implementations in nvCVLoggerExamples.c:

MemLogger

Collects logs into a memory string buffer

StderrLogger

Logs directly to stderr

FileLogger

Logs to a single file

FileThreadLogger

Logs to file in separate thread for better performance

MultifileLogger

Logs to multiple files in rotation

4.2.27.4 Usage Examples

Basic Logging

```
// Log to stderr
NvAR_ConfigureLogger(LOG_ERROR, "stderr", NULL, NULL);

// Log to file
NvAR_ConfigureLogger(LOG_ERROR, "/logs/ar.log", NULL, NULL);

// Disable logging
NvAR_ConfigureLogger(LOG_ERROR, NULL, NULL, NULL);
```

Custom Logger Examples

```
// Memory logger
MemLogger memLog;
NvAR_ConfigureLogger(LOG_ERROR, NULL, &MemLogger::Callback, &memLog);

// File logger with append
FileLogger flog("/path/to/ar.log", "a");
NvAR_ConfigureLogger(LOG_ERROR, NULL, &FileLogger::Callback, &flog);

// Multi-file logger (12 files, 64KB each)
MultifileLogger mflog("/path/to/log%02d.log", 65536, 12, 0);
NvAR_ConfigureLogger(LOG_ERROR, NULL, &MultifileLogger::Callback, &mflog);
```

Dynamic Verbosity

```
// Save and restore verbosity
int saved_verbosity;
NvAR_ConfigureLogger(LOG_INFO, "same", NULL, &saved_verbosity);
// ... code logged at INFO level ...
NvAR_ConfigureLogger(saved_verbosity, "same", NULL, NULL);
```

4.2.27.5 Remarks

1. The StderrLogger and FileLogger are compiled into the SDK for easy configuration
2. FileThreadLogger reduces performance impact compared to standard FileLogger

3. MultifileLogger is useful for long-running services to prevent unbounded log growth
4. Custom loggers can be implemented to suit specific application needs
5. Use “same” as the file parameter to change verbosity while keeping the same logger configuration

4.2.28. NvAR_ResetState

```
NvAR_Result NvAR_ResetState(
    NvAR_FeatureHandle handle,
    nvAR_StateHandle state
);
```

4.2.28.1 Parameters

handle [in]

Type: NvAR_FeatureHandle
The handle to the feature instance to be reset.

state [in]

Type: NvAR_StateHandle
The handle to the state object to be reset (nullptr for default state).

4.2.28.2 Return Value

Returns one of the following values:

- NVCV_SUCCESS on success
- NVCV_ERR_MISSINGINPUT
- NVCV_ERR_FEATURENOTFOUND
- NVCV_ERR_UNIMPLEMENTED

4.2.28.3 Remarks

This function resets the input state object of the feature instance with the specified handle. When the state object handle is set to nullptr, the default state is reset. This function is only implemented for the Eye Contact feature and returns NVCV_ERR_UNIMPLEMENTED error for other features.

4.3. Return Codes

The `NvCV_Status` enumeration defines the following values that the NVIDIA AR SDK functions might return to indicate error or success:

NVCV_SUCCESS = 0

Success: The function executed successfully.

NVCV_ERR_GENERAL

Generic error: The function failed to execute for an unspecified reason.

NVCV_ERR_UNIMPLEMENTED

The requested feature is not implemented.

NVCV_ERR_MEMORY

Memory error: The requested operation requires more memory than is available.

NVCV_ERR_EFFECT

Invalid effect handle has been supplied.

NVCV_ERR_SELECTOR

The specified selector is not valid for this effect filter.

NVCV_ERR_BUFFER

No image buffer has been specified.

NVCV_ERR_PARAMETER

An invalid parameter value has been supplied for this combination of effect and selector string.

NVCV_ERR_MISMATCH

Some parameters, for example, image formats or image dimensions, are not correctly matched.

NVCV_ERR_PIXELFORMAT

The specified pixel format is not supported.

NVCV_ERR_MODEL

An error occurred while the TRT model was being loaded.

NVCV_ERR_LIBRARY

An error occurred while the dynamic library was being loaded.

NVCV_ERR_INITIALIZATION

The effect has not been properly initialized.

NVCV_ERR_FILE

The specified file could not be found.

NVCV_ERR_FEATURENOTFOUND

The requested feature was not found.

NVCV_ERR_MISSINGINPUT

A required parameter was not set.

NVCV_ERR_RESOLUTION

The specified image resolution is not supported.

NVCV_ERR_UNSUPPORTEDGPU

The GPU is not supported.

NVCV_ERR_WRONGGPU

The current GPU is not the one selected.

NVCV_ERR_UNSUPPORTEDDRIVER

The currently installed graphics driver is not supported.

NVCV_ERR_MODELDEPENDENCIES

There is no model with dependencies that match this system.

NVCV_ERR_PARSE

There has been a parsing or syntax error while reading a file.

NVCV_ERR_MODELSUBSTITUTION

The specified model does not exist and has been substituted.

NVCV_ERR_READ

An error occurred while reading a file.

NVCV_ERR_WRITE

An error occurred while writing a file.

NVCV_ERR_PARAMREADONLY

The selected parameter is read-only.

NVCV_ERR_TRT_ENQUEUE

TensorRT enqueue failed.

NVCV_ERR_TRT_BINDINGS

Unexpected TensorRT bindings.

NVCV_ERR_TRT_CONTEXT

An error occurred while creating a TensorRT context.

NVCV_ERR_TRT_INFER

There was a problem creating the inference engine.

NVCV_ERR_TRT_ENGINE

There was a problem deserializing the inference runtime engine.

NVCV_ERR_NPP

An error has occurred in the NPP library.

NVCV_ERR_CONFIG

No suitable model exists for the specified parameter configuration.

NVCV_ERR_TOOSMALL

The supplied parameter or buffer is not large enough.

NVCV_ERR_TOOBIG

The supplied parameter is too big.

NVCV_ERR_WRONGSIZE

The supplied parameter is not the expected size.

NVCV_ERR_OBJECTNOTFOUND

The specified object was not found.

NVCV_ERR_SINGULAR

A mathematical singularity has been encountered.

NVCV_ERR_NOTHINGRENDERED

Nothing was rendered in the specified region.

NVCV_ERR_OPENGL

An OpenGL error has occurred.

NVCV_ERR_DIRECT3D

A Direct3D error has occurred.

NVCV_ERR_CUDA_MEMORY

The requested operation requires more CUDA memory than is available.

NVCV_ERR_CUDA_VALUE

A CUDA parameter is not within its acceptable range.

NVCV_ERR_CUDA_PITCH

A CUDA pitch is not within its acceptable range.

NVCV_ERR_CUDA_INIT

The CUDA driver and runtime could not be initialized.

NVCV_ERR_CUDA_LAUNCH

The CUDA kernel failed to launch.

NVCV_ERR_CUDA_KERNEL

No suitable kernel image is available for the device.

NVCV_ERR_CUDA_DRIVER

The installed NVIDIA CUDA driver is older than the CUDA runtime library.

NVCV_ERR_CUDA_UNSUPPORTED

The CUDA operation is not supported on the current system or device.

NVCV_ERR_CUDA_ILLEGAL_ADDRESS

CUDA attempted to load or store an invalid memory address.

NVCV_ERR_CUDA

An unspecified CUDA error has occurred.

Note

Many other CUDA-related errors exist that are not listed here. However, you can use the function `NvCV_GetErrorStringFromCode()` to convert the error code into a string to help you debug.

Chapter 5. Appendix A: NVIDIA 3DMM File Format

The NVIDIA 3DMM file format is based on encapsulated objects that are scoped by a FOURCC tag and a 32-bit size.

The header must appear first in the file. The objects and their subobjects can appear in any order. In this guide, they are listed in the default order.

5.1. Header

The header contains the following information:

- The name NFAC
- size=8
- endian=0xe4 (little endian)
- sizeBits=32
- indexBits=16
- The offset of the table of contents

NFAC				
size				
	endian	sizeBits	indexBits	zero
	TOC loc			

5.2. Model Object

The model object contains a shape component and an optional color component. Both objects contain the following information:

- A mean shape
- A set of shape modes

- The eigenvalues for the modes
- A triangle list

** MODL**			
size			
	SHAP		
	size		
		MEAN	
		size	
			mean shape
		BSIS	
		size	
			number of modes
			shape modes
			...
		EIVL	
		size	
			shape eigenvalues
		TRNG	
		size	
			triangle list
	COLR		
	size		
		MEAN	
		size	
			mean color
		BSIS	
		size	
			number of modes
			color modes
			...
		EIVL	
		size	
			color eigenvalues

continues on next page

Table 1 – continued from previous page

** MODL**		
	TRNG	
	size	
		triangle list

5.3. IBUG Mappings Object

The IBUG mappings object contains the following information:

- Landmarks
- Right contour
- Left contour

IBUG		
size		
	LMRK	
	size	
		landmarks
	RCTR	
	size	
		right contour
	LCTR	
	size	
		left contour

5.4. Blend Shapes Object

The blend shapes object contains a set of blend shapes, and each blend shape has a name.

BLND		
size		
	numShapes	
	NAME	
	size	
		name string
	SHAP	
	size	
		blend shape
	NAME	
	size	
		name string
	SHAP	
	size	
		blend shape
		...

5.5. Model Contours Object

The model contours object contains a right contour and a left contour.

MCTR		
size		
	RCTR	
	size	
		right model contour
	LCTR	
	size	
		left model contour

5.6. Topology Object

The topology contains a list of pairs of the adjacent faces and vertices.

TOPO		
size		
	AJFC	
	size	
		adjacent faces
	AJVX	
	size	
		adjacent vertices

5.7. NVIDIA Landmarks Object

NVIDIA expands the number of landmarks from 68 to 126, including more detailed contours on the left and right.

NVLM		
size		
	LMRK	
	size	
		Landmarks
	RCTR	
	size	
		Right contour
	LCTR	
	size	
		Left contour

5.8. Partition Object

This partitions the mesh into coherent submeshes of the same material, used for rendering.

PRTS			
size			
–	**PART**		
	size		
		Partition index	
		Face index	
		Number of faces	
		Vertex index	
		Number of vertices	
		Smoothing group	
		NAME	
		size	
			Partition name string
		MTRL	
		size	
			Material name string
	PART	(any number of additional partitions)	
	...		

5.9. Table of Contents Object

The optional table of contents object contains a list of tagged objects and their offsets. This object can be used to randomly access objects. The file is usually read in sequential order.

TOC0	
size	
	record size
	tag
	offset
	tag
	offset
	...

Chapter 6. Appendix B: 3D Body Pose Keypoint Format

6.1. Legacy 3D Body Pose Tracking: 34-Keypoint Format

The legacy 3D Body Pose Tracking feature uses 34 keypoints: pelvis, left hip, right hip, torso, left knee, right knee, neck, left ankle, right ankle, left big toe, right big toe, left small toe, right small toe, left heel, right heel, nose, left eye, right eye, left ear, right ear, left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left pinky knuckle, right pinky knuckle, left middle tip, right middle tip, left index knuckle, right index knuckle, left thumb tip, right thumb tip.

Here is the list of skeletal structures:

Keypoint	Parent
Pelvis	None, as this is root.
Left hip	Pelvis
Right hip	Pelvis
Torso	Pelvis
Left knee	Left hip
Right knee	Right hip
Neck	Torso
Left ankle	Left knee
Right ankle	Right knee
Left big toe	Left ankle
Right big toe	Right ankle
Left small toe	Left ankle
Right small toe	Right ankle
Left heel	Left ankle
Right heel	Right ankle

continues on next page

Table 1 – continued from previous page

Keypoint	Parent
Nose	Neck
Left eye	Nose
Right eye	Nose
Left ear	Nose
Right ear	Nose
Left shoulder	Neck
Right shoulder	Neck
Left elbow	Left shoulder
Right elbow	Right shoulder
Left wrist	Left elbow
Right wrist	Right elbow
Left pinky knuckle	Left wrist
Right pinky knuckle	Right wrist
Left middle tip	Left wrist
Right middle tip	Right wrist
Left index knuckle	Left wrist
Right index knuckle	Right wrist
Left thumb tip	Left wrist
Right thumb tip	Right wrist

6.2. NvAR_Parameter_Output KeyPoints Order

In the macros `NvAR_Parameter_Output(KeyPoints)` and `NvAR_Parameter_Output(KeyPoints3D)`, the order of the keypoints is the same as listed in *Legacy 3D Body Pose Tracking: 34-Keypoint Format*.

6.3. 3D Body Pose Estimation SOMA-77 Format

3D Body Pose Estimation uses the SOMA-77 public skeleton. Before you allocate output buffers, query `NvAR_Parameter_Config(NumKeyPoints)` to get the keypoint count.

The underlying full-body SOMA skeleton has 78 joints, including an internal virtual Root joint at index 0. The AR SDK exposes the 77 user-facing joints and omits the virtual Root joint. Output keypoint and pose index `k` maps to internal SOMA joint `k + 1`, so output index `0` is the Hips joint.

The public SOMA-X rig assets are available from NVIDIA's SOMA-X Hugging Face repository. That repository includes the standard rig definition and related assets, including `SOMA_template_rig.usda`, `SOMA_neutral.npz`, `SOMA_procedural_transforms.json`, and `correctives_model.pt`.

6.3.1. RestPose and T-Pose Bases

3D Body Pose Estimation reports animation outputs in two related bases. A basis is the joint-frame convention used by local animation data: the frame in which each parent-to-child rest offset is expressed, and the frame that each local joint rotation is relative to. It is not a camera or world coordinate system, and it is not one global rotation that can be applied independently to every accumulated joint point.

- The **direct-rest basis** is the SDK's native output basis. For joint `j`, the usable local rest offset is `RestPose[j] - RestPose[parent(j)]`. That offset is expressed in `parent(j)`'s direct-rest frame. `JointRotations` are local rotations in the same direct-rest hierarchy. Running forward kinematics over those local offsets and rotations reconstructs root-relative posed 3D points; applying `RootPose` places those points in camera coordinates.
- The **T-pose basis** is the joint-frame convention used by the SOMA-X template rig's visual T-pose. Rigs authored against a T-pose expect parent-to-child offsets and local rotations in this basis. To move from the SDK direct-rest basis to the T-pose basis, each local rest offset is rotated by the fixed SOMA-X parent T-pose-basis joint orientation, and each local rotation is converted with the corresponding parent and child SOMA-X T-pose-basis joint orientations.

The `RestPose` output uses cumulative direct-rest storage: joint 0 is at the origin, and each non-root joint stores its parent's `RestPose` value plus that joint's local rest offset in the parent direct-rest frame. Therefore, plotting `RestPose` directly does not produce the visual SOMA-X T-pose. *Reconstructing T-Pose and Posed 3D Keypoints* shows how to convert the direct-rest local offsets and rotations into the T-pose skeleton and T-pose-basis rotations.

The `KeyPoints`, `KeyPoints3D`, `RestPose`, `JointRotations`, and `JointAngles` outputs use the same body-major, keypoint-major layout:

```
body_index * NumKeyPoints + keypoint_index
```

`KeyPoints` uses full-image pixel coordinates. For 3D Body Pose Estimation, `KeyPoints3D` uses camera coordinates in meters. `RestPose` is in meters and uses the cumulative direct-rest representation. `JointRotations` are local XYZW quaternions in the same direct-rest basis. `JointAngles` is a compatibility alias for `JointRotations`. `RootPose` stores one root transform per body; `RootPose.translation` is in camera coordinates, in meters.

6.3.2. SOMA-77 Parent Indices

The Hips keypoint has the parent -1. All other parent values are keypoint indices in the same output order.

```
index:  0  parent: -1
index:  1  parent:  0
index:  2  parent:  1
index:  3  parent:  2
index:  4  parent:  3
index:  5  parent:  4
index:  6  parent:  5
index:  7  parent:  6
index:  8  parent:  6
index:  9  parent:  6
index: 10  parent:  6
index: 11  parent:  3
index: 12  parent: 11
```

(continues on next page)

(continued from previous page)

```
index: 13  parent: 12
index: 14  parent: 13
index: 15  parent: 14
index: 16  parent: 15
index: 17  parent: 16
index: 18  parent: 17
index: 19  parent: 14
index: 20  parent: 19
index: 21  parent: 20
index: 22  parent: 21
index: 23  parent: 22
index: 24  parent: 14
index: 25  parent: 24
index: 26  parent: 25
index: 27  parent: 26
index: 28  parent: 27
index: 29  parent: 14
index: 30  parent: 29
index: 31  parent: 30
index: 32  parent: 31
index: 33  parent: 32
index: 34  parent: 14
index: 35  parent: 34
index: 36  parent: 35
index: 37  parent: 36
index: 38  parent: 37
index: 39  parent:  3
index: 40  parent: 39
index: 41  parent: 40
index: 42  parent: 41
index: 43  parent: 42
index: 44  parent: 43
index: 45  parent: 44
index: 46  parent: 45
index: 47  parent: 42
index: 48  parent: 47
index: 49  parent: 48
index: 50  parent: 49
index: 51  parent: 50
index: 52  parent: 42
index: 53  parent: 52
index: 54  parent: 53
index: 55  parent: 54
index: 56  parent: 55
index: 57  parent: 42
index: 58  parent: 57
index: 59  parent: 58
index: 60  parent: 59
index: 61  parent: 60
index: 62  parent: 42
index: 63  parent: 62
index: 64  parent: 63
```

(continues on next page)

(continued from previous page)

```
index: 65  parent: 64
index: 66  parent: 65
index: 67  parent:  0
index: 68  parent: 67
index: 69  parent: 68
index: 70  parent: 69
index: 71  parent: 70
index: 72  parent:  0
index: 73  parent: 72
index: 74  parent: 73
index: 75  parent: 74
index: 76  parent: 75
```

6.3.3. Reconstructing T-Pose and Posed 3D Keypoints

The `RestPose`, `JointRotations`, and `RootPose` outputs are enough to reconstruct the emitted `KeyPoints3D` points for the SOMA-77 skeleton. You can also derive person-specific skeleton geometry in the T-pose basis that is useful for retargeting.

Use the output buffers from the same delayed output frame, after `NvAR_Parameter_Output(Ready)` becomes nonzero. The example uses these SDK outputs:

- `rest_pose_output` is the buffer registered with `NvAR_Parameter_Output(RestPose)`.
- `joint_rotations_output` is the buffer registered with `NvAR_Parameter_Output(JointRotations)`.
- `root_pose_output` is the buffer registered with `NvAR_Parameter_Output(RootPose)`.

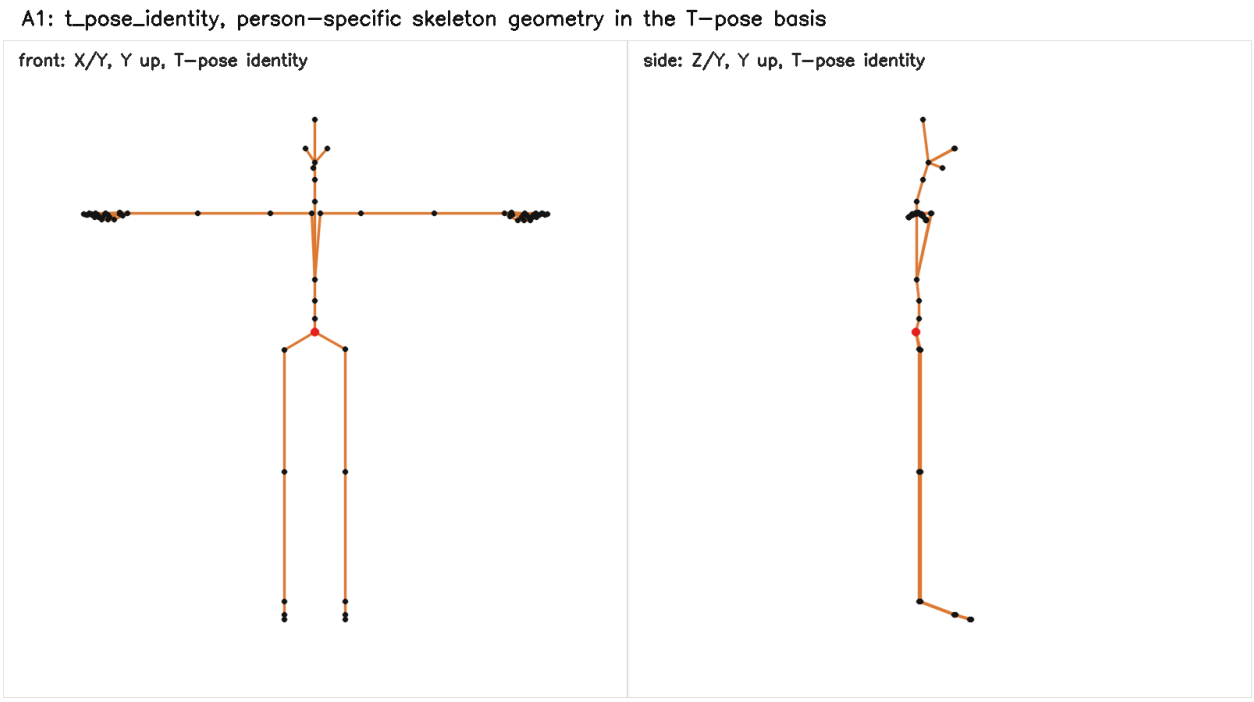
`KeyPoints3D` is not an input to the reconstruction. Use it only to validate the reconstructed camera-space points.

The example derives these values:

- `t_pose_identity` is the person-specific skeleton geometry in the T-pose basis. It is a root-relative SOMA-77 point set in meters, derived from `RestPose` and the SOMA-X T-pose-basis joint-orientation table.
- `t_pose_local_rotations` are local joint rotations converted from `JointRotations` into the T-pose basis. Apply these rotations to a rig that uses `t_pose_identity` as its bind skeleton.
- `retarget_local` is the posed root-relative SOMA-77 skeleton produced by running forward kinematics over `t_pose_identity` with `t_pose_local_rotations`.
- `retarget_in_camera` is `retarget_local` after applying the T-pose-basis root rotation and `RootPose.translation`. It should match `KeyPoints3D`.

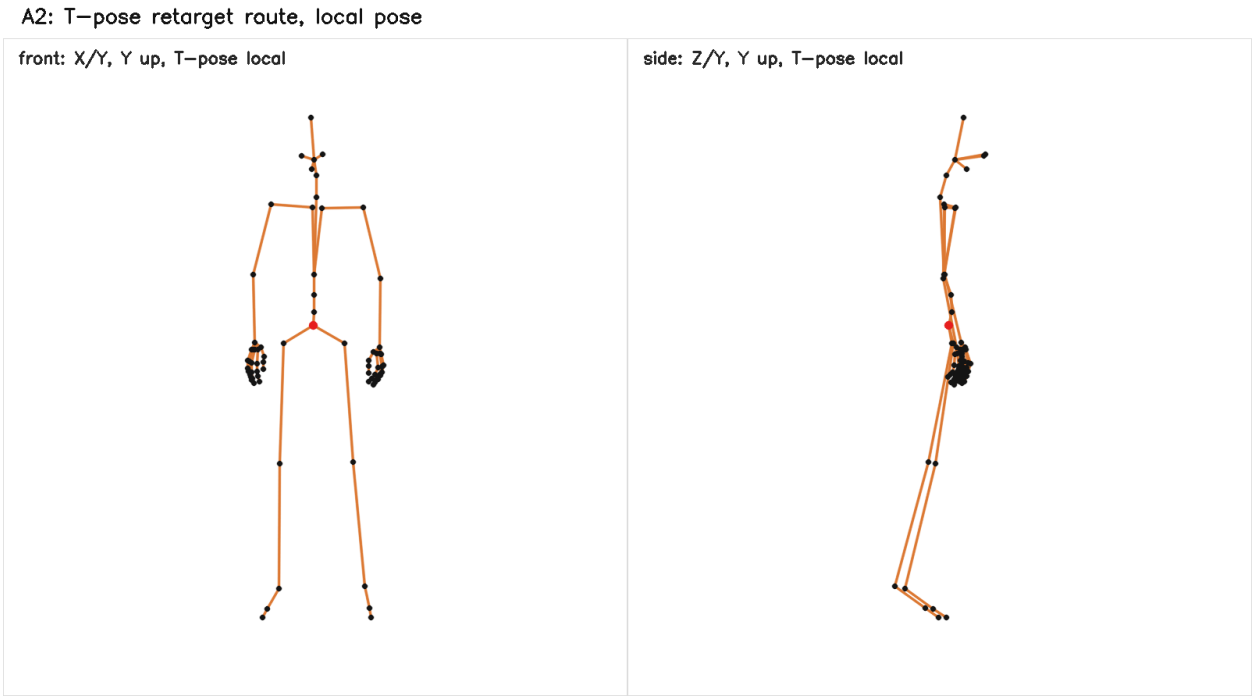
The retargeting reconstruction has three steps.

A1: Derive t_pose_identity from the rest pose. Use `RestPose` and the SOMA-X T-pose-basis joint-orientation table to derive a root-relative point set in meters. This step converts the SDK direct-rest local offsets into person-specific skeleton geometry in the T-pose basis. A retargeting rig can use `t_pose_identity` as its bind geometry.



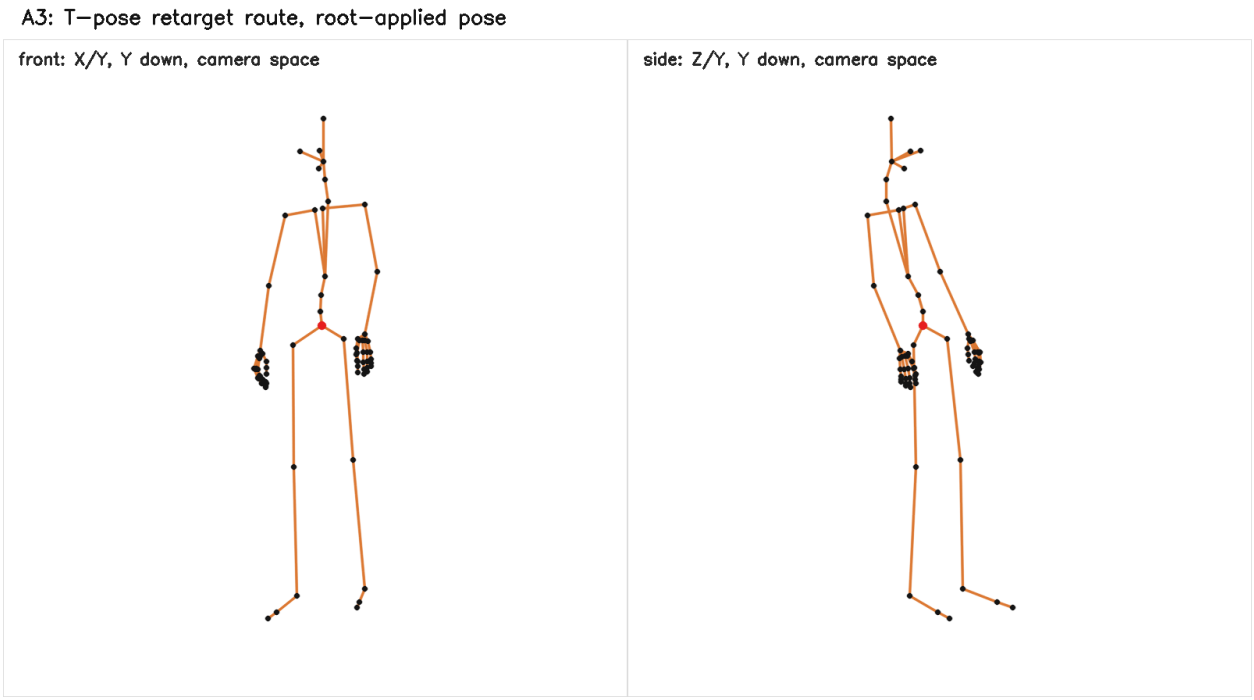
A1: t_pose_identity derived from the rest pose.

A2: Reconstruct the posed skeleton in the T-pose basis. First convert `JointRotations` into `t_pose_local_rotations`, which are local pose rotations in the T-pose basis. Then run forward kinematics over A1's `t_pose_identity` using `t_pose_local_rotations`. The result is `retarget_local`, a posed root-relative SOMA-77 skeleton in the T-pose basis. Use this output to drive a retargeted T-pose rig.



A2: Posed skeleton reconstructed locally in the T-pose basis.

A3: Apply root placement. Apply the T-pose-basis root rotation and `RootPose.translation` to `retarget_local`. The result is `retarget_in_camera`, the retargeted pose in camera coordinates.



A3: Posed skeleton with the camera-space root translation and T-pose-basis root rotation applied.

The following example keeps A1, A2, and A3 in one function because A2 reuses A1's derived T-pose, and all three steps share the same skeleton mapping and math helpers. It also includes a direct reconstruction path that reconstructs `KeyPoints3D` through the SDK direct-rest basis for validation and debugging.

The code assumes these application-local math helpers exist: `AddPoint`, `SubtractPoint`, `RotatePoint`, `RotatePointByMatrix3`, `NormalizeQuaternion`, `QuaternionMultiply`, `InvertQuaternion`, and `MatrixToQuaternion`.

```
#include <cstddef>
#include <cstdint>
#include <vector>

constexpr std::uint32_t kNumKp = 77u;

static constexpr std::int32_t kSoma77Parents[kNumKp] = {
    -1, 0,  1,  2,  3,  4,  5,  6,  6,  6,  6,  3,  11, 12, 13, 14, 15, 16,
↪17, 14, 19, 20, 21, 22, 14, 24,
    25, 26, 27, 14, 29, 30, 31, 32, 14, 34, 35, 36, 37, 3,  39, 40, 41, 42,
↪43, 44, 45, 42, 47, 48, 49, 50,
    42, 52, 53, 54, 55, 42, 57, 58, 59, 60, 42, 62, 63, 64, 65, 0,  67, 68,
↪69, 70, 0,  72, 73, 74, 75};
static constexpr std::size_t kSomaInternalJointCount = kNumKp + 1u;
static constexpr std::size_t kSomaTPoseBasisJointOrientMatrixDim = 9u;

// Fixed row-major 3x3 T-pose-basis joint-orientation matrices for the internal
↪SOMA skeleton.
```

(continues on next page)

(continued from previous page)

```
// These constants match the SDK SOMA model joint orientations. Corresponding
 ↪public SOMA-X rig assets are
// available at https://huggingface.co/nvidia/SOMA-X/tree/main.
// Public joint j maps to internal joint j + 1. Internal joint 0 is the virtual
 ↪Root.
static const float kSomaTPoseBasisJointOrient[kSomaInternalJointCount *
 ↪kSomaTPoseBasisJointOrientMatrixDim] = {
      1.0000000000e+00f,  0.0000000000e+00f,  0.0000000000e+00f,  0.
↪0000000000e+00f,  1.0000000000e+00f,
      0.0000000000e+00f,  0.0000000000e+00f,  0.0000000000e+00f,  1.
↪0000000000e+00f,  3.2073343981e-10f,
      1.6653345369e-16f,  1.0000000000e+00f,  9.9999445677e-01f,  3.
↪3239081968e-03f,  -3.2073166345e-10f,
      -3.3239081968e-03f, 9.9999445677e-01f,  1.0660361482e-12f,  -2.
↪5863775122e-08f, -5.2050825872e-08f,
      1.0000000000e+00f,  9.9999123812e-01f,  4.1855261661e-03f,  2.
↪6081409032e-08f,  -4.1855261661e-03f,
      9.9999123812e-01f,  5.1942116386e-08f,  -7.5423216117e-08f, -5.
↪4443827224e-08f, 1.0000000000e+00f,
      9.9421060085e-01f,  1.0744910687e-01f,  8.0836500160e-08f,  -1.
↪0744910687e-01f, 9.9421060085e-01f,
      4.6024471345e-08f,  -1.7273029584e-07f, -4.6310127289e-08f, 1.
↪0000000000e+00f,  9.9999374151e-01f,
      3.5373084247e-03f,  1.7289302434e-07f,  -3.5373084247e-03f, 9.
↪9999374151e-01f,  4.5698836715e-08f,
      -3.5810501231e-07f, 6.2969554904e-08f,  1.0000000000e+00f,  9.
↪5817577839e-01f,  -2.8618034720e-01f,
      3.6114818158e-07f,  2.8618034720e-01f,  9.5817577839e-01f,  4.
↪2146709234e-08f,  -7.2769233839e-07f,
      6.9529072277e-08f,  1.0000000000e+00f,  9.5276397467e-01f,  -3.
↪0371171236e-01f, 7.1443582783e-07f,
      3.0371171236e-01f,  9.5276397467e-01f,  1.5476389592e-07f,  -1.
↪4150831475e-06f, -5.9895546656e-07f,
      1.0000000000e+00f,  9.9307656288e-01f,  1.1746867001e-01f,  1.
↪4756444671e-06f,  -1.1746867001e-01f,
      9.9307656288e-01f,  4.2858070515e-07f,  -7.9456022206e-08f, -8.
↪5877069012e-08f, 1.0000000000e+00f,
      9.9307656288e-01f,  1.1746867001e-01f,  8.8993779457e-08f,  -1.
↪1746867001e-01f, 9.9307656288e-01f,
      7.5948918266e-08f,  -2.8270396797e-06f, -5.9895546656e-07f, 1.
↪0000000000e+00f,  9.9307656288e-01f,
      1.1746867001e-01f,  2.8778254091e-06f,  -1.1746867001e-01f, 9.
↪9307656288e-01f,  2.6272005016e-07f,
      4.4457851909e-03f,  6.6037088633e-02f,  9.9780726433e-01f,  9.
↪9773859978e-01f,  -6.7214086652e-02f,
      2.8974004636e-06f,  6.7066892982e-02f,  9.9555081129e-01f,  -6.
↪6186569631e-02f, -4.4516143389e-03f,
      -6.6037207842e-02f, 9.9780720472e-01f,  9.9773854017e-01f,  -6.
↪7214466631e-02f, 2.8974004636e-06f,
      6.7066892982e-02f,  9.9555075169e-01f,  6.6187083721e-02f,  9.
↪3822997808e-01f,  3.4601223469e-01f,
      -3.4477467619e-16f, -4.3630720053e-16f, 3.8786763620e-17f,  -1.
```

(continues on next page)

(continued from previous page)

```
↪0000000000e+00f, -3.4601223469e-01f,
      9.3822997808e-01f,  1.8670427278e-16f,  1.0000000000e+00f,  9.
↪0094777988e-05f,  -3.4477467619e-16f,
      -3.9595269803e-16f, 1.8732287153e-16f,  -1.0000000000e+00f, -9.
↪0094777988e-05f, 1.0000000000e+00f,
      1.8670427278e-16f,  1.0000000000e+00f,  -9.6392890555e-05f, -3.
↪4477467619e-16f, -3.9591775793e-16f,
      1.8739670898e-16f,  -1.0000000000e+00f, 9.6392890555e-05f,  1.
↪0000000000e+00f,  1.8670427278e-16f,
      1.0000000000e+00f,  -1.0125053086e-04f, 5.2388550102e-05f,  3.
↪4557331674e-05f,  -1.6869449615e-01f,
      -9.8566836119e-01f, 1.0863710486e-04f,  9.8566836119e-01f,  -1.
↪6869449615e-01f, 8.5283172131e-01f,
      -4.9016791582e-01f, 1.8003733456e-01f,  -3.8852432370e-01f, -8.
↪2598888874e-01f, -4.0840080380e-01f,
      3.4889379144e-01f,  2.7834826708e-01f,  -8.9487171173e-01f, 1.
↪0000000000e+00f,  5.5511151231e-16f,
      2.4240717877e-08f,  5.1347814889e-16f,  -1.0000000000e+00f, -3.
↪3306690739e-16f, 2.4240717877e-08f,
      4.9960036108e-16f,  -1.0000000000e+00f, 1.0000000000e+00f,  5.
↪5511151231e-16f,  2.4240717877e-08f,
      5.1347814889e-16f,  -1.0000000000e+00f, -3.3306690739e-16f, 2.
↪4240717877e-08f,  4.9960036108e-16f,
      -1.0000000000e+00f, 1.0000000000e+00f,  5.5511151231e-16f,  2.
↪4240717877e-08f,  5.1347814889e-16f,
      -1.0000000000e+00f, -3.3306690739e-16f, 2.4240717877e-08f,  4.
↪9960036108e-16f,  -1.0000000000e+00f,
      9.9960470200e-01f,  -2.7482686564e-02f, 5.9245699085e-03f,  1.
↪8942133756e-03f,  -1.4441768825e-01f,
      -9.8951500654e-01f, 2.8050143272e-02f,  9.8913508654e-01f,  -1.
↪4430855215e-01f, 1.0000000000e+00f,
      -2.1739474221e-08f, 8.6736173799e-19f,  -5.2041704279e-17f, -2.
↪7755575616e-16f, -1.0000000000e+00f,
      2.1739474221e-08f,  1.0000000000e+00f,  -3.0531133177e-16f, 1.
↪0000000000e+00f,  -2.1739474221e-08f,
      8.6736173799e-19f,  -5.2041704279e-17f, -2.7755575616e-16f, -1.
↪0000000000e+00f, 2.1739474221e-08f,
      1.0000000000e+00f,  -3.0531133177e-16f, 9.9703454971e-01f,  4.
↪3452162296e-02f,  -6.3514046371e-02f,
      -6.5227262676e-02f, 3.9215739816e-02f,  -9.9709957838e-01f, -4.
↪0835380554e-02f, 9.9828553200e-01f,
      4.1933711618e-02f,  9.9703454971e-01f,  4.3452162296e-02f,  -6.
↪3514046371e-02f, -6.5227262676e-02f,
      3.9215739816e-02f,  -9.9709957838e-01f, -4.0835380554e-02f, 9.
↪9828553200e-01f,  4.1933711618e-02f,
      9.8629713058e-01f,  1.5425087512e-01f,  -5.8520063758e-02f, -4.
↪1307274252e-02f, -1.1252379417e-01f,
      -9.9279004335e-01f, -1.5972362459e-01f, 9.8160332441e-01f,  -1.
↪0461021215e-01f, 1.0000000000e+00f,
      -2.3722368070e-08f, 4.1633363423e-17f,  -4.1633363423e-17f, -1.
↪5265566589e-16f, -1.0000000000e+00f,
      2.3722368070e-08f,  1.0000000000e+00f,  -1.9428902931e-16f, 1.
```

(continues on next page)

(continued from previous page)

```
↪0000000000e+00f,  -2.3722368070e-08f,
      4.1633363423e-17f,  -4.1633363423e-17f, -1.5265566589e-16f, -1.
↪0000000000e+00f, 2.3722368070e-08f,
      1.0000000000e+00f,  -1.9428902931e-16f, 9.9183505774e-01f,  2.
↪1237600595e-02f,  -1.2574636936e-01f,
      -1.2679313123e-01f, 5.8578822762e-02f,  -9.9019795656e-01f, -1.
↪3663354330e-02f, 9.9805682898e-01f,
      6.0793314129e-02f,  9.9183505774e-01f,  2.1237600595e-02f,  -1.
↪2574636936e-01f, -1.2679313123e-01f,
      5.8578822762e-02f,  -9.9019795656e-01f, -1.3663354330e-02f, 9.
↪9805682898e-01f,  6.0793314129e-02f,
      9.7038769722e-01f,  2.1955496073e-01f,  -1.0071407259e-01f, -8.
↪0587044358e-02f, -9.8797477782e-02f,
      -9.9183911085e-01f, -2.2771349549e-01f, 9.7058469057e-01f,  -7.
↪8178569674e-02f, 1.0000000000e+00f,
      -2.3967489327e-08f, -1.3877787808e-17f, -5.5511151231e-17f, -9.
↪7144514655e-17f, -1.0000000000e+00f,
      2.3967489327e-08f,  1.0000000000e+00f,  -1.3877787808e-16f, 1.
↪0000000000e+00f,  -2.3967489327e-08f,
      -1.3877787808e-17f, -5.5511151231e-17f, -9.7144514655e-17f, -1.
↪0000000000e+00f, 2.3967489327e-08f,
      1.0000000000e+00f,  -1.3877787808e-16f, 9.9919605255e-01f,  7.
↪9067231854e-04f,  4.0083222091e-02f,
      4.0091000497e-02f,  -1.8781829625e-02f, -9.9901950359e-01f, -3.
↪7060817704e-05f, 9.9982327223e-01f,
      -1.8798429519e-02f, 9.9919605255e-01f,  7.9067231854e-04f,  4.
↪0083222091e-02f,  4.0091000497e-02f,
      -1.8781829625e-02f, -9.9901950359e-01f, -3.7060817704e-05f, 9.
↪9982327223e-01f,  -1.8798429519e-02f,
      9.1683477163e-01f,  3.1319043040e-01f,  -2.4764028192e-01f, -2.
↪3987418413e-01f, -6.3751161098e-02f,
      -9.6870851517e-01f, -3.1917759776e-01f, 9.4754815102e-01f,  1.
↪6677020118e-02f,  1.0000000000e+00f,
      -2.3872162913e-08f, 8.3266726847e-17f,  -2.7755575616e-17f, -1.
↪9775847626e-16f, -1.0000000000e+00f,
      2.3872162913e-08f,  1.0000000000e+00f,  -2.6714741530e-16f, 1.
↪0000000000e+00f,  -2.3872162913e-08f,
      8.3266726847e-17f,  -2.7755575616e-17f, -1.9775847626e-16f, -1.
↪0000000000e+00f, 2.3872162913e-08f,
      1.0000000000e+00f,  -2.6714741530e-16f, 9.9629634619e-01f,  -2.
↪6847388595e-02f, -8.1687375903e-02f,
      -8.0836147070e-02f, 3.1395409256e-02f,  -9.9623280764e-01f, 2.
↪9310859740e-02f,  9.9914640188e-01f,
      2.9108893126e-02f,  9.9629634619e-01f,  -2.6847388595e-02f, -8.
↪1687375903e-02f, -8.0836147070e-02f,
      3.1395409256e-02f,  -9.9623280764e-01f, 2.9310859740e-02f,  9.
↪9914640188e-01f,  2.9108893126e-02f,
      9.3822997808e-01f,  3.4601223469e-01f,  3.4477467619e-16f,  -2.
↪0335961397e-16f, -5.4376201197e-16f,
      1.0000000000e+00f,  3.4601223469e-01f,  -9.3822997808e-01f, -4.
↪6426002894e-16f, 1.0000000000e+00f,
      9.0240624559e-05f,  3.4477467619e-16f,  -3.7890671584e-16f, -4.
```

(continues on next page)

(continued from previous page)

```
↪3984308663e-16f, 1.0000000000e+00f,
      9.0240624559e-05f,  -1.0000000000e+00f, -4.6426002894e-16f, 1.
↪0000000000e+00f,  -9.6549476439e-05f,
      3.4477467619e-16f,  -3.7898885156e-16f, -4.3977230643e-16f, 1.
↪0000000000e+00f,  -9.6549476439e-05f,
      -1.0000000000e+00f, -4.6426002894e-16f, 1.0000000000e+00f,  -1.
↪0125053086e-04f, 5.2388550102e-05f,
      -3.4557331674e-05f, 1.6869449615e-01f,  9.8566836119e-01f,  -1.
↪0863710486e-04f, -9.8566836119e-01f,
      1.6869449615e-01f,  8.5284388065e-01f,  -4.9015244842e-01f, 1.
↪8002195656e-01f,  3.8852506876e-01f,
      8.2600259781e-01f,  4.0837234259e-01f,  -3.4886330366e-01f, -2.
↪7833479643e-01f, 8.9488774538e-01f,
      1.0000000000e+00f,  1.3877787808e-16f,  -2.4239820817e-08f, -1.
↪3877787808e-17f, 1.0000000000e+00f,
      3.3306690739e-16f,  2.4239820817e-08f,  -1.6653345369e-16f, 1.
↪0000000000e+00f,  1.0000000000e+00f,
      1.3877787808e-16f,  -2.4239820817e-08f, -1.3877787808e-17f, 1.
↪0000000000e+00f,  3.3306690739e-16f,
      2.4239820817e-08f,  -1.6653345369e-16f, 1.0000000000e+00f,  1.
↪0000000000e+00f,  1.3877787808e-16f,
      -2.4239820817e-08f, -1.3877787808e-17f, 1.0000000000e+00f,  3.
↪3306690739e-16f,  2.4239820817e-08f,
      -1.6653345369e-16f, 1.0000000000e+00f,  9.9960327148e-01f,  -2.
↪7519376948e-02f, 6.0034482740e-03f,
      -1.9657290541e-03f, 1.4446231723e-01f,  9.8950833082e-01f,  -2.
↪8097925708e-02f, -9.8912757635e-01f,
      1.4435090125e-01f,  1.0000000000e+00f,  2.1738612688e-08f,  4.
↪3368086899e-18f,  -3.9898639947e-17f,
      -2.7755575616e-17f, 1.0000000000e+00f,  2.1738612688e-08f,  -1.
↪0000000000e+00f, 2.7755575616e-17f,
      1.0000000000e+00f,  2.1738612688e-08f,  4.3368086899e-18f,  -3.
↪9898639947e-17f, -2.7755575616e-17f,
      1.0000000000e+00f,  2.1738612688e-08f,  -1.0000000000e+00f, 2.
↪7755575616e-17f,  9.9703550339e-01f,
      4.3437462300e-02f,  -6.3508957624e-02f, 6.5221473575e-02f,  -3.
↪9213620126e-02f, 9.9709999561e-01f,
      4.0821075439e-02f,  -9.9828624725e-01f, -4.1930425912e-02f, 9.
↪9703550339e-01f,  4.3437462300e-02f,
      -6.3508957624e-02f, 6.5221473575e-02f,  -3.9213620126e-02f, 9.
↪9709999561e-01f,  4.0821075439e-02f,
      -9.9828624725e-01f, -4.1930425912e-02f, 9.8629844189e-01f,  1.
↪5424472094e-01f,  -5.8514416218e-02f,
      4.1302621365e-02f,  1.1252149940e-01f,  9.9279052019e-01f,  1.
↪5971682966e-01f,  -9.8160451651e-01f,
      1.0460907221e-01f,  1.0000000000e+00f,  2.3721467457e-08f,  6.
↪9388939039e-18f,  -3.4694469520e-17f,
      -5.4123372450e-16f, 1.0000000000e+00f,  2.3721467457e-08f,  -1.
↪0000000000e+00f, -4.7184478547e-16f,
      1.0000000000e+00f,  2.3721467457e-08f,  6.9388939039e-18f,  -3.
↪4694469520e-17f, -5.4123372450e-16f,
      1.0000000000e+00f,  2.3721467457e-08f,  -1.0000000000e+00f, -4.
```

(continues on next page)

(continued from previous page)

```
↪7184478547e-16f, 9.9183410406e-01f,
      2.1235136315e-02f,  -1.2575449049e-01f, 1.2680117786e-01f,  -5.
↪8585133404e-02f, 9.9019658566e-01f,
      1.3659616001e-02f,  -9.9805653095e-01f, -6.0799375176e-02f, 9.
↪9183410406e-01f,  2.1235136315e-02f,
      -1.2575449049e-01f, 1.2680117786e-01f,  -5.8585133404e-02f, 9.
↪9019658566e-01f,  1.3659616001e-02f,
      -9.9805653095e-01f, -6.0799375176e-02f, 9.7038930655e-01f,  2.
↪1955129504e-01f,  -1.0070653260e-01f,
      8.0579929054e-02f,  9.8796248436e-02f,  9.9183976650e-01f,  2.
↪2770914435e-01f,  -9.7058564425e-01f,
      7.8179396689e-02f,  1.0000000000e+00f,  2.3966544305e-08f,  -2.
↪7755575616e-17f, 4.1633363423e-17f,
      -1.8041124150e-16f, 1.0000000000e+00f,  2.3966544305e-08f,  -1.
↪0000000000e+00f, -8.3266726847e-17f,
      1.0000000000e+00f,  2.3966544305e-08f,  -2.7755575616e-17f, 4.
↪1633363423e-17f,  -1.8041124150e-16f,
      1.0000000000e+00f,  2.3966544305e-08f,  -1.0000000000e+00f, -8.
↪3266726847e-17f, 9.9919724464e-01f,
      7.7960011549e-04f,  4.0053173900e-02f,  -4.0060751140e-02f, 1.
↪8767835572e-02f,  9.9902099371e-01f,
      2.7125513952e-05f,  -9.9982357025e-01f, 1.8783999607e-02f,  9.
↪9919724464e-01f,  7.7960011549e-04f,
      4.0053173900e-02f,  -4.0060751140e-02f, 1.8767835572e-02f,  9.
↪9902099371e-01f,  2.7125513952e-05f,
      -9.9982357025e-01f, 1.8783999607e-02f,  9.1683918238e-01f,  3.
↪1317952275e-01f,  -2.4763785303e-01f,
      2.3987253010e-01f,  6.3747830689e-02f,  9.6870911121e-01f,  3.
↪1916621327e-01f,  -9.4755202532e-01f,
      -1.6676655039e-02f, 1.0000000000e+00f,  2.3871425725e-08f,  3.
↪6082248300e-16f,  -2.7755575616e-17f,
      -1.5265566589e-16f, 1.0000000000e+00f,  2.3871425725e-08f,  -1.
↪0000000000e+00f, -4.8572257327e-17f,
      1.0000000000e+00f,  2.3871425725e-08f,  3.6082248300e-16f,  -2.
↪7755575616e-17f, -1.5265566589e-16f,
      1.0000000000e+00f,  2.3871425725e-08f,  -1.0000000000e+00f, -4.
↪8572257327e-17f, 9.9630093575e-01f,
      -2.6843685657e-02f, -8.1632398069e-02f, 8.0781921744e-02f,  -3.
↪1373634934e-02f, 9.9623793364e-01f,
      -2.9303802177e-02f, -9.9914717674e-01f, -2.9089098796e-02f, 9.
↪9630093575e-01f,  -2.6843685657e-02f,
      -8.1632398069e-02f, 8.0781921744e-02f,  -3.1373634934e-02f, 9.
↪9623793364e-01f,  -2.9303802177e-02f,
      -9.9914717674e-01f, -2.9089098796e-02f, -3.2073169121e-10f, 1.
↪0623073601e-12f,  1.0000000000e+00f,
      -1.0000000000e+00f, -1.1269920833e-05f, -3.2073166345e-10f, 1.
↪1269920833e-05f,  -1.0000000000e+00f,
      1.0660361482e-12f,  -3.2073166345e-10f, 1.0696111965e-12f,  1.
↪0000000000e+00f,  -1.0000000000e+00f,
      1.1502694178e-05f,  -3.2073166345e-10f, -1.1502694178e-05f, -1.
↪0000000000e+00f, 1.0660361482e-12f,
      -1.1554545515e-10f, -2.9919758338e-10f, 1.0000000000e+00f,  -3.
```

(continues on next page)

(continued from previous page)

```
↪5715162754e-01f, -9.3404644728e-01f,
      -3.2073166345e-10f, 9.3404644728e-01f,  -3.5715162754e-01f, 1.
↪0660361482e-12f,  -1.4164447784e-03f,
      4.1140656685e-04f,  9.9999892712e-01f,  -2.7892348170e-01f, -9.
↪6031332016e-01f, -4.8847298428e-10f,
      9.6031230688e-01f,  -2.7892318368e-01f, 1.4749816619e-03f,  -1.
↪4234089758e-03f, 3.8662354928e-04f,
      9.9999892712e-01f,  -2.6212123036e-01f, -9.6503496170e-01f, -4.
↪8847298428e-10f, 9.6503388882e-01f,
      -2.6212093234e-01f, 1.4749816619e-03f,  3.2073169121e-10f,  -1.
↪0623074686e-12f, 1.0000000000e+00f,
      1.0000000000e+00f,  1.1269637980e-05f,  -3.2073166345e-10f, -1.
↪1269637980e-05f, 1.0000000000e+00f,
      1.0660361482e-12f,  3.2073166345e-10f,  -1.0696111965e-12f, 1.
↪0000000000e+00f,  1.0000000000e+00f,
      -1.1502432244e-05f, -3.2073166345e-10f, 1.1502432244e-05f,  1.
↪0000000000e+00f,  1.0660361482e-12f,
      1.1554545515e-10f,  2.9919758338e-10f,  1.0000000000e+00f,  3.
↪5715162754e-01f,  9.3404644728e-01f,
      -3.2073166345e-10f, -9.3404644728e-01f, 3.5715162754e-01f,  1.
↪0660361482e-12f,  -1.4170479262e-03f,
      4.1158948443e-04f,  9.9999892712e-01f,  2.7892661095e-01f,  9.
↪6031242609e-01f,  -2.1203838685e-09f,
      -9.6031135321e-01f, 2.7892631292e-01f,  -1.4756119344e-03f, -1.
↪4240153832e-03f, 3.8679590216e-04f,
      9.9999892712e-01f,  2.6212435961e-01f,  9.6503412724e-01f,  -2.
↪1203838685e-09f, -9.6503305435e-01f,
      2.6212409139e-01f,  -1.4756119344e-03f
};

static void ReconstructSoma77PoseFromSdkOutputs(const NvAR_Point3f* rest_pose_
↪output,
                                                const NvAR_Quaternion* joint_
↪rotations_output,
                                                const NvAR_Transform3f* root_
↪pose_output, std::size_t body_index,
                                                std::vector<NvAR_Point3f>* t_
↪pose_identity_output,
                                                std::vector<NvAR_Point3f>*
↪retarget_local_output,
                                                std::vector<NvAR_Point3f>*
↪retarget_in_camera_output,
                                                std::vector<NvAR_Point3f>*
↪direct_reconstruction_local_output,
                                                std::vector<NvAR_Point3f>*
↪direct_reconstruction_root_applied_output) {
  // These pointers must refer to the delayed output frame where NvAR_
↪Parameter_Output(Ready) is nonzero:
  //
  //   rest_pose_output       -> NvAR_Parameter_Output(RestPose)
  //   joint_rotations_output -> NvAR_Parameter_Output(JointRotations)
  //   root_pose_output       -> NvAR_Parameter_Output(RootPose)
```

(continues on next page)

(continued from previous page)

```
  //
  // NvAR_Parameter_Output(KeyPoints3D) is not required for reconstruction.
↪Use it only to validate that the
  // reconstructed camera-space points match the SDK-emitted keypoints.
  const std::size_t body_offset = body_index * static_cast<std::size_t>
↪(kNumKp);
  std::vector<NvAR_Point3f> rest_pose(rest_pose_output + body_offset, rest_
↪pose_output + body_offset + kNumKp);
  std::vector<NvAR_Quaternion> joint_rotations(joint_rotations_output + body_
↪offset,
                                               joint_rotations_output + body_
↪offset + kNumKp);
  const NvAR_Transform3f root_pose = root_pose_output[body_index];

  const auto to_internal_soma_joint = [](std::size_t public_joint) { return
↪public_joint + 1u; };
  const auto to_internal_soma_parent = [&to_internal_soma_joint](std::size_t
↪public_joint) {
    const std::int32_t parent = kSoma77Parents[public_joint];
    return parent < 0 ? 0u : to_internal_soma_joint(static_cast<std::size_t>
↪(parent));
  };
  const auto soma_t_pose_basis_joint_orient = [](std::size_t internal_joint) {
    return kSomaTPoseBasisJointOrient + internal_joint *
↪kSomaTPoseBasisJointOrientMatrixDim;
  };

  // A1. Derive t_pose_identity from RestPose.
  // RestPose stores a cumulative direct-rest representation: joint 0 is at
↪the origin, and every non-root joint is
  // stored as its parent's RestPose value plus that joint's local rest
↪translation expressed in the parent direct-rest
  // frame. Plotting RestPose directly does not produce the visual SOMA T-pose.
↪ To derive t_pose_identity, the
  // person-specific skeleton geometry in the T-pose basis, recover each local
↪rest translation with
  // RestPose[joint] - RestPose[parent], rotate it by the fixed parent SOMA T-
↪pose-basis joint-orientation matrix, and
  // accumulate the transformed local translations.
  std::vector<NvAR_Point3f> t_pose_identity(kNumKp, NvAR_Point3f{0.0f, 0.0f,
↪0.0f});
  for (std::size_t joint = 1u; joint < kNumKp; ++joint) {
    const std::size_t parent = static_cast<std::size_t>
↪(kSoma77Parents[joint]);
    const NvAR_Point3f rest_local_translation = SubtractPoint(rest_
↪pose[joint], rest_pose[parent]);
    const NvAR_Point3f t_pose_local_translation =
        RotatePointByMatrix3(soma_t_pose_basis_joint_orient(to_internal_soma_
↪joint(parent)),
                             rest_local_translation);
    t_pose_identity[joint] = AddPoint(t_pose_identity[parent], t_pose_local_
↪translation);
```

(continues on next page)

(continued from previous page)

```
  }

  // A2. Reconstruct the posed skeleton in the T-pose basis.
  // Convert each JointRotation into t_pose_local_rotations, local pose
↪rotations in the T-pose basis:
  //
  //   t_pose_local_rotation[j] =
  //       t_pose_basis_joint_orient[parent(j)] * JointRotations[j] *
↪inverse(t_pose_basis_joint_orient[j])
  //
  // Then run FK over the t_pose_identity local translations using t_pose_
↪local_rotations. retarget_local is the posed
  // skeleton in the T-pose basis, before root placement.
  std::vector<NvAR_Quaternion> t_pose_local_rotations(kNumKp, NvAR_Quaternion
↪{0.0f, 0.0f, 0.0f, 1.0f});
  for (std::size_t joint = 0u; joint < kNumKp; ++joint) {
    const NvAR_Quaternion parent_orient =
        MatrixToQuaternion(soma_t_pose_basis_joint_orient(to_internal_soma_
↪parent(joint)));
    const NvAR_Quaternion joint_orient =
        MatrixToQuaternion(soma_t_pose_basis_joint_orient(to_internal_soma_
↪joint(joint)));
    t_pose_local_rotations[joint] = NormalizeQuaternion(
        QuaternionMultiply(QuaternionMultiply(parent_orient,
↪NormalizeQuaternion(joint_rotations[joint])),
                           InvertQuaternion(joint_orient)));
  }
  std::vector<NvAR_Point3f> retarget_local(kNumKp, NvAR_Point3f{0.0f, 0.0f, 0.
↪0f});
  std::vector<NvAR_Quaternion> retarget_global_rotations(kNumKp, NvAR_
↪Quaternion{0.0f, 0.0f, 0.0f, 1.0f});
  for (std::size_t joint = 1u; joint < kNumKp; ++joint) {
    const std::size_t parent = static_cast<std::size_t>
↪(kSoma77Parents[joint]);
    const NvAR_Point3f t_pose_local_translation =
        SubtractPoint(t_pose_identity[joint], t_pose_identity[parent]);
    retarget_local[joint] =
        AddPoint(retarget_local[parent], RotatePoint(retarget_global_
↪rotations[parent], t_pose_local_translation));
    retarget_global_rotations[joint] =
        NormalizeQuaternion(QuaternionMultiply(retarget_global_
↪rotations[parent], t_pose_local_rotations[joint]));
  }

  // A3. Apply root placement to the T-pose retargeting route.
  // RootPose.rotation and JointRotations[0] carry the same direct-rest-basis
↪root rotation in SDK output. The T-pose
  // route derives the T-pose-basis equivalent of the root rotation from
↪JointRotations[0]. Applying that converted root
  // rotation and RootPose.translation places retarget_local in camera
↪coordinates.
  std::vector<NvAR_Point3f> retarget_in_camera(kNumKp, NvAR_Point3f{0.0f, 0.
```

(continues on next page)

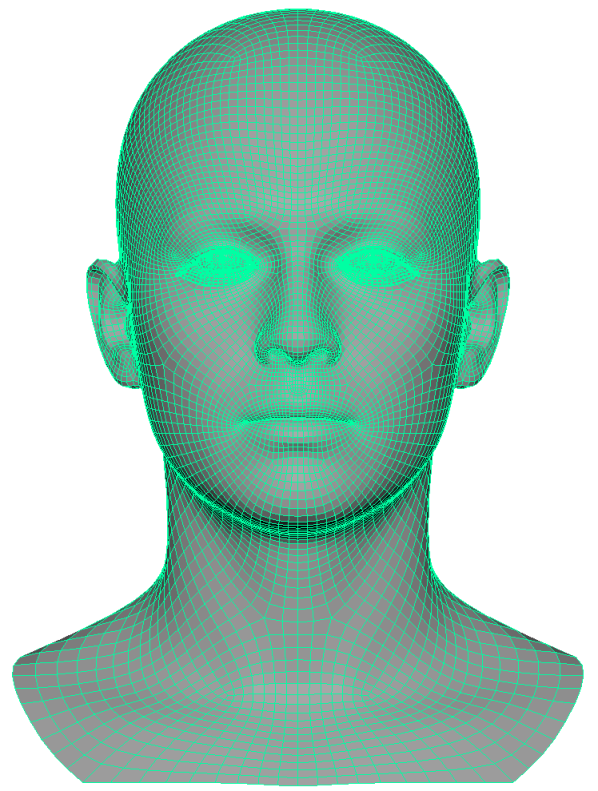
(continued from previous page)

```
↪0f, 0.0f});
  for (std::size_t joint = 0u; joint < kNumKp; ++joint) {
    retarget_in_camera[joint] =
        AddPoint(root_pose.translation, RotatePoint(t_pose_local_rotations[0],
↪ retarget_local[joint]));
  }

  // B. Directly reconstruct the emitted KeyPoints3D through the SDK direct-
↪rest basis.
  // This route is useful for validation and debugging. It is not the
↪preferred retargeting route because the
  // local point set is in the direct-rest basis, not the T-pose basis.
  std::vector<NvAR_Point3f> direct_reconstruction_local(kNumKp, NvAR_Point3f
↪{0.0f, 0.0f, 0.0f});
  std::vector<NvAR_Quaternion> direct_reconstruction_global_rotations(kNumKp,
                                                                      NvAR_
↪Quaternion{0.0f, 0.0f, 0.0f, 1.0f});
  for (std::size_t joint = 1u; joint < kNumKp; ++joint) {
    const std::size_t parent = static_cast<std::size_t>
↪(kSoma77Parents[joint]);
    const NvAR_Point3f rest_local_translation = SubtractPoint(rest_
↪pose[joint], rest_pose[parent]);
    direct_reconstruction_local[joint] = AddPoint(
        direct_reconstruction_local[parent],
        RotatePoint(direct_reconstruction_global_rotations[parent], rest_
↪local_translation));
    direct_reconstruction_global_rotations[joint] = NormalizeQuaternion(
        QuaternionMultiply(direct_reconstruction_global_rotations[parent],
↪joint_rotations[joint]));
  }
  std::vector<NvAR_Point3f> direct_reconstruction_root_applied(kNumKp, NvAR_
↪Point3f{0.0f, 0.0f, 0.0f});
  for (std::size_t joint = 0u; joint < kNumKp; ++joint) {
    direct_reconstruction_root_applied[joint] =
        AddPoint(root_pose.translation, RotatePoint(root_pose.rotation,
↪direct_reconstruction_local[joint]));
  }

  *t_pose_identity_output = t_pose_identity;
  *retarget_local_output = retarget_local;
  *retarget_in_camera_output = retarget_in_camera;
  *direct_reconstruction_local_output = direct_reconstruction_local;
  *direct_reconstruction_root_applied_output = direct_reconstruction_root_
↪applied;
}
```

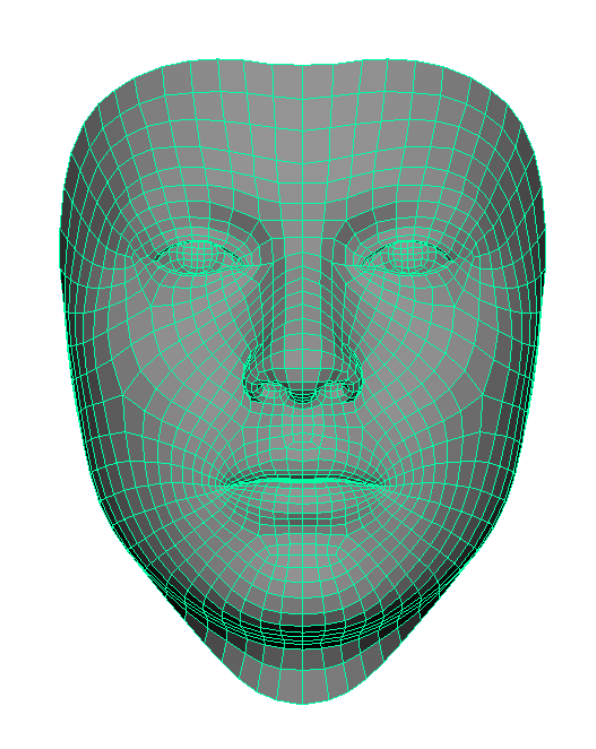
Chapter 7. Appendix C: 3DMM Versions

The SDK includes a default face model that is used by the face fitting feature. This model is a modification of the ICT Face Model from the ICT-FaceKit repository. The modified version of the face model, `face_model2.nvf`, is optimized for real-time face fitting applications and has a lower resolution. In addition to the blendshapes provided by the ICT Model, the model uses linear blendshapes for eye gaze expressions, which enables the model to be used in implicit gaze tracking.

Accompanying `face_model2.nvf` is a `face_model2.mtl` file, which is associated with the Wavefront OBJ file format and describes the photometric materials that are used only when rendering `face_model2.nvf` in the ExpressionApp sample application. The `face_model2.mtl` text file can be edited to change the appearance and is not used for mesh or expression fitting.



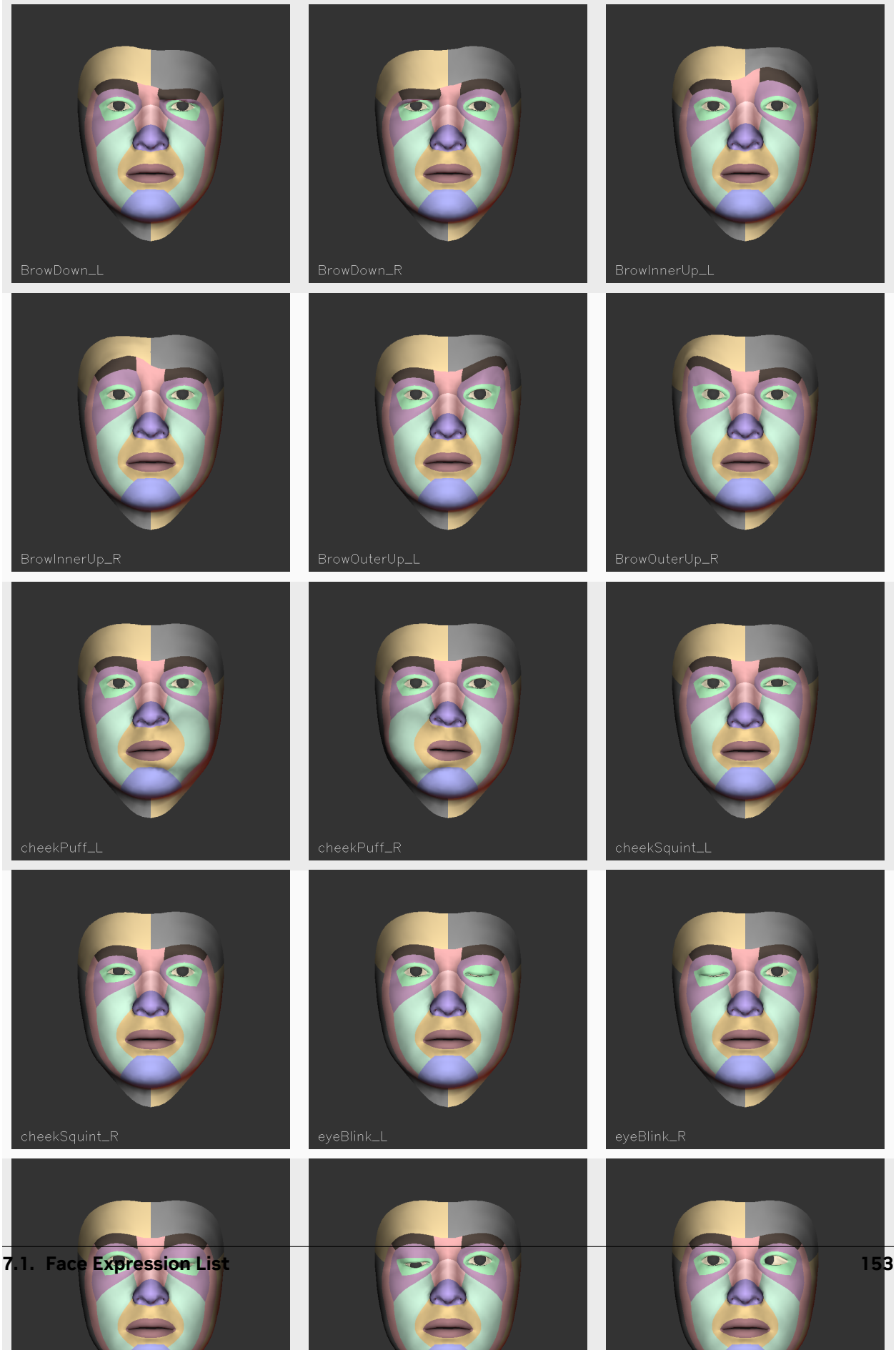
The original ICT face model topology



The modified face model topology of `face_model2.nvf`

7.1. Face Expression List

The following images show all expression blendshapes that are used in the face expression estimation feature.


BrowDown_L
BrowDown_R
BrowInnerUp_L
BrowInnerUp_R
BrowOuterUp_L
BrowOuterUp_R
cheekPuff_L
cheekPuff_R
cheekSquint_L
cheekSquint_R
eyeBlink_L
eyeBlink_R

Here is the face blendshape expression list that is used in the Face 3D Mesh feature and the Facial Expression Estimation feature:

- 0: `BrowDown_L`
- 1: `BrowDown_R`
- 2: `BrowInnerUp_L`
- 3: `BrowInnerUp_R`
- 4: `BrowOuterUp_L`
- 5: `BrowOuterUp_R`
- 6: `cheekPuff_L`
- 7: `cheekPuff_R`
- 8: `cheekSquint_L`
- 9: `cheekSquint_R`
- 10: `eyeBlink_L`
- 11: `eyeBlink_R`
- 12: `eyeLookDown_L`
- 13: `eyeLookDown_R`
- 14: `eyeLookIn_L`
- 15: `eyeLookIn_R`
- 16: `eyeLookOut_L`
- 17: `eyeLookOut_R`
- 18: `eyeLookUp_L`
- 19: `eyeLookUp_R`
- 20: `eyeSquint_L`
- 21: `eyeSquint_R`
- 22: `eyeWide_L`
- 23: `eyeWide_R`
- 24: `jawForward`
- 25: `jawLeft`
- 26: `jawOpen`
- 27: `jawRight`
- 28: `mouthClose`
- 29: `mouthDimple_L`
- 30: `mouthDimple_R`
- 31: `mouthFrown_L`
- 32: `mouthFrown_R`
- 33: `mouthFunnel`

- 34: `mouthLeft`
- 35: `mouthLowerDown_L`
- 36: `mouthLowerDown_R`
- 37: `mouthPress_L`
- 38: `mouthPress_R`
- 39: `mouthPucker`
- 40: `mouthRight`
- 41: `mouthRollLower`
- 42: `mouthRollUpper`
- 43: `mouthShrugLower`
- 44: `mouthShrugUpper`
- 45: `mouthSmile_L`
- 46: `mouthSmile_R`
- 47: `mouthStretch_L`
- 48: `mouthStretch_R`
- 49: `mouthUpperUp_L`
- 50: `mouthUpperUp_R`
- 51: `noseSneer_L`
- 52: `noseSneer_R`

The items in the preceding list can be mapped to the ARKit blendshapes by using the following options:

- `A01_Brow_Inner_Up = 0.5 * (browInnerUp_L + browInnerUp_R)`
- `A02_Brow_Down_Left = browDown_L`
- `A03_Brow_Down_Right = browDown_R`
- `A04_Brow_Outer_Up_Left = browOuterUp_L`
- `A05_Brow_Outer_Up_Right = browOuterUp_R`
- `A06_Eye_Look_Up_Left = eyeLookUp_L`
- `A07_Eye_Look_Up_Right = eyeLookUp_R`
- `A08_Eye_Look_Down_Left = eyeLookDown_L`
- `A09_Eye_Look_Down_Right = eyeLookDown_R`
- `A10_Eye_Look_Out_Left = eyeLookOut_L`
- `A11_Eye_Look_In_Left = eyeLookIn_L`
- `A12_Eye_Look_In_Right = eyeLookIn_R`
- `A13_Eye_Look_Out_Right = eyeLookOut_R`
- `A14_Eye_Blink_Left = eyeBlink_L`
- `A15_Eye_Blink_Right = eyeBlink_R`
- `A16_Eye_Squint_Left = eyeSquint_L`

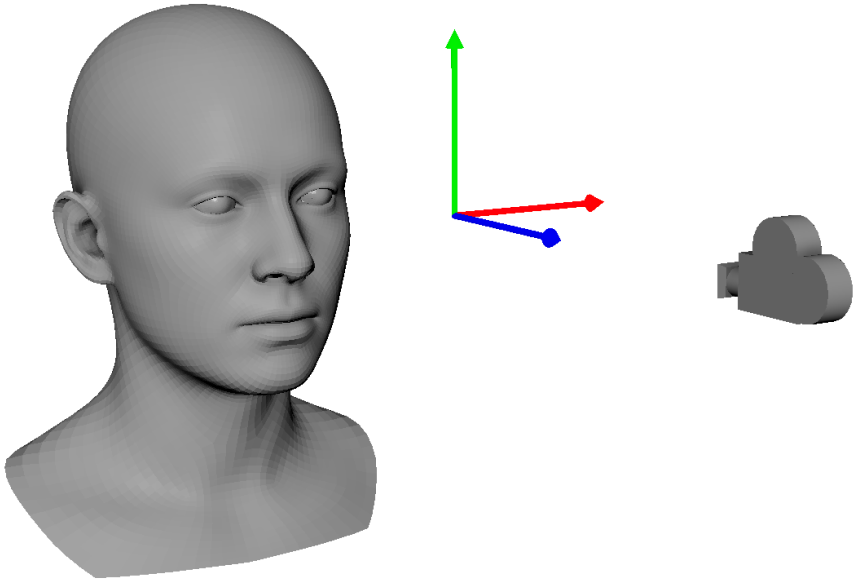
- A17_Eye_Squint_Right = eyeSquint_R
- A18_Eye_Wide_Left = eyeWide_L
- A19_Eye_Wide_Right = eyeWide_R
- A20_Cheek_Puff = 0.5 * (cheekPuff_L + cheekPuff_R)
- A21_Cheek_Squint_Left = cheekSquint_L
- A22_Cheek_Squint_Right = cheekSquint_R
- A23_Nose_Sneer_Left = noseSneer_L
- A24_Nose_Sneer_Right = noseSneer_R
- A25_Jaw_Open = jawOpen
- A26_Jaw_Forward = jawForward
- A27_Jaw_Left = jawLeft
- A28_Jaw_Right = jawRight
- A29_Mouth_Funnel = mouthFunnel
- A30_Mouth_Pucker = mouthPucker
- A31_Mouth_Left = mouthLeft
- A32_Mouth_Right = mouthRight
- A33_Mouth_Roll_Upper = mouthRollUpper
- A34_Mouth_Roll_Lower = mouthRollLower
- A35_Mouth_Shrug_Upper = mouthShrugUpper
- A36_Mouth_Shrug_Lower = mouthShrugLower
- A37_Mouth_Close = mouthClose
- A38_Mouth_Smile_Left = mouthSmile_L
- A39_Mouth_Smile_Right = mouthSmile_R
- A40_Mouth_Frown_Left = mouthFrown_L
- A41_Mouth_Frown_Right = mouthFrown_R
- A42_Mouth_Dimple_Left = mouthDimple_L
- A43_Mouth_Dimple_Right = mouthDimple_R
- A44_Mouth_Upper_Up_Left = mouthUpperUp_L
- A45_Mouth_Upper_Up_Right = mouthUpperUp_R
- A46_Mouth_Lower_Down_Left = mouthLowerDown_L
- A47_Mouth_Lower_Down_Right = mouthLowerDown_R
- A48_Mouth_Press_Left = mouthPress_L
- A49_Mouth_Press_Right = mouthPress_R
- A50_Mouth_Stretch_Left = mouthStretch_L
- A51_Mouth_Stretch_Right = mouthStretch_R
- A52_Tongue_Out = 0

Chapter 8. Appendix D: Coordinate Systems

The documentation may refer to different coordinate systems where virtual 3D objects are defined. This appendix defines the different coordinate spaces that can be interfaced with through the SDK.

8.1. NvAR World 3D Space

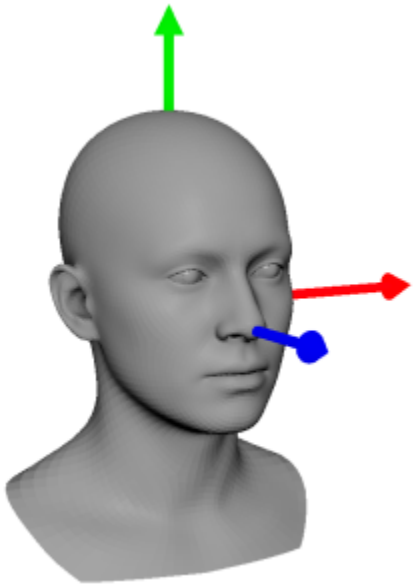
NvAR World 3D space, or simply *world space*, defines the main reference frame used in 3D applications. The reference frame is right handed, and the coordinate units are centimeters. The camera is usually, but not always, placed so that it is looking down the negative z-axis, with the x-axis pointing right, and the y-axis pointing up. The face model is usually, but not always placed so that the positive z-axis points out from the nose of the model, the x-axis points out from the left ear, and the y-axis points up from the head. This is so that no rotation of a face model or a camera model would correspond to the model being in view.



NvAR World 3D space

8.2. NvAR Model 3D Space

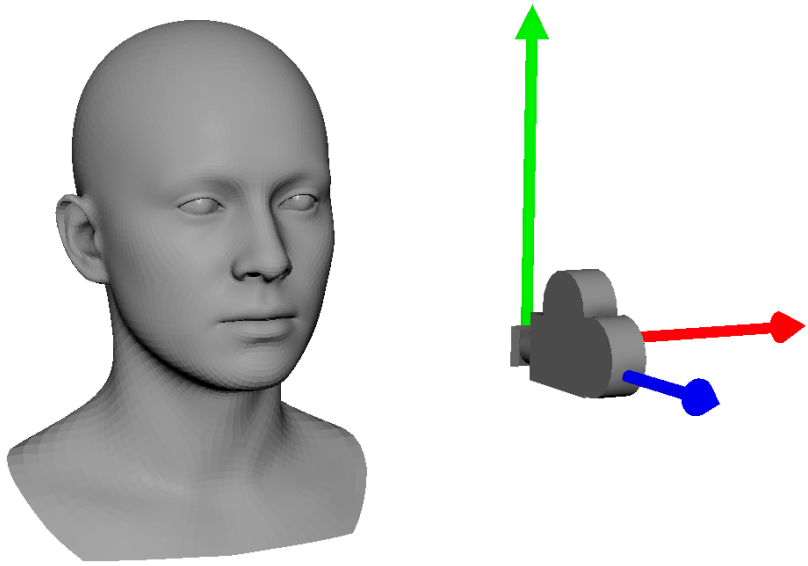
In NvAR Model 3D space, or simply the *model space*, a face model has its origin in the center of the skull where the first neck joint would be placed. The reference frame is right handed, and the coordinate units are centimeters. The positive z-axis points out from the nose of the model, the x-axis points out from the left ear, and the y-axis points up from the head.



NvAR Model 3D space

8.3. NvAR Camera 3D Space

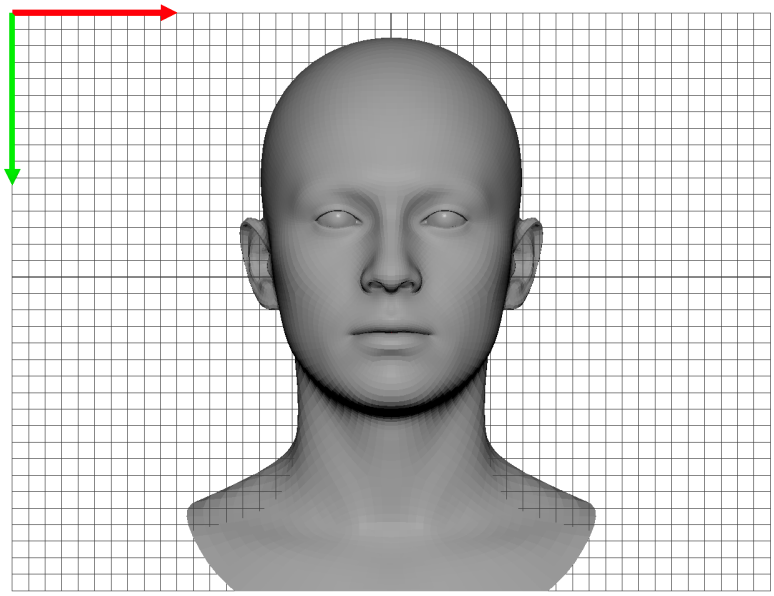
In NvAR Camera 3D space, or simply the *camera space*, objects are placed in relation to a specific camera. The camera space can in many cases be the same as the world space unless explicit camera extrinsics are used. The reference frame is right handed, and the coordinate units are centimeters. The positive x-axis points to the right, the y-axis points up and the z-axis points back.



NvAR Camera 3D space

8.4. NvAR Image 2D Space

The NvAR Image 2D space, or *screen space*, is the main 2D space for screen coordinates. The origin of an image is the location of the uppermost, leftmost pixel; the x-axis points to the right and the y-axis points down. The unit of this coordinate system is pixels.

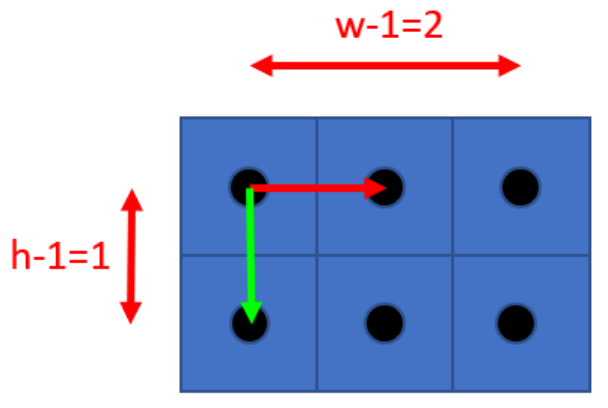


NvAR Image 2D space

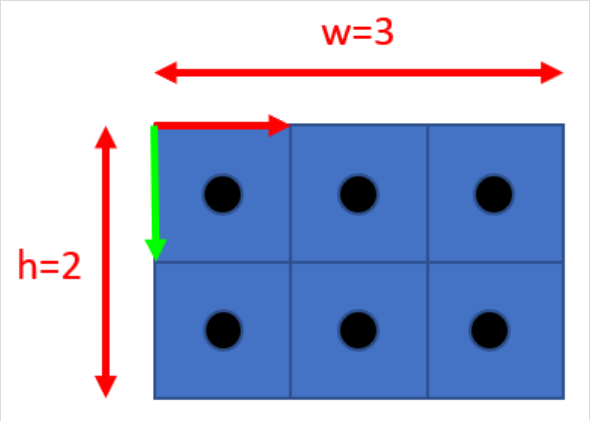
This coordinate system has two flavors: *on-pixel* or *inter-pixel*.

In the on-pixel system, the origin point is located on the center of the pixel. In the inter-pixel system, the origin is offset from the center of the pixel by a distance of −0.5 pixels along the x-axis and the y-axis.

On-pixel should be used for integer-based coordinates; inter-pixel should be used for real-valued coordinates (usually defined as floating-point coordinates).



On-pixel integer coordinates



Inter-pixel real-valued coordinates

Copyright