Sample Code

Creating a 3D application with hydra rendering

Build a 3D application that integrates with Hydra and USD.

Download

Overview

Note

This sample code project is associated with WWDC22 session 10141: Explore USD tools and rendering.

Configure the sample code project

This sample requires Xcode 14 or later and macOS 13 or later. To build the project, you first need to get and build the USD source code from the Pixar GitHub repository, and then use CMake to generate an Xcode project with references to both the compiled USD libraries and the header files in the USD source code. If you don't already have CMake installed, download the latest version of CMake to your Applications folder.

CMake is both a GUI and command-line app. To use the command-line tool, open a Terminal window and add the `/Contents/bin` folder from the `CMake.app` application bundle to your PATH environment variable, like this:

```
path+=('/Applications/CMake.app/Contents/bin/')
export PATH
```

Note

The previous command assumes you use the default `zsh` shell and adds `cmake` to your path for only the current terminal session. To add `cmake` to your path permanently, or if you're using another shell like `bash`, add `/Applications/CMake.app/Contents/bin/` to the $PATH declaration in your `.zshrc` file or in the configuration file your shell uses.

Clone the USD repo, using the following command:

```
git clone https://github.com/PixarAnimationStudios/USD
```

Next, build USD using the following command: `python3 <path to usd source>/build_scripts/build_usd.py --generator Xcode --no-python <path to install the built USD>`. For example, if you've cloned the USD source code into `~/dev/USD`, the build command might look like this:

```
python3 ~/dev/USD/build_scripts/build_usd.py --generator Xcode --no-python ./USDInst
```

Configure the USD_`Path` environment variable: `export USD_PATH=<path to usd install>`. For example, if you've installed USD at `~/dev/USDInstall`, use this command:

```
 export USD_PATH=~/dev/USDInstall
```

Run the following CMake command to generate an Xcode project: `cmake -S <path to project source folder> -B <path to directory where it creates the Xcode project>`. If the sample code is at `~/dev/`, the command might look like this:

```
cmake -S ~/dev/CreatingA3DApplicationWithHydraRendering/ -B ~/dev/CreatingA3DApplica
```

Finally, open the generated Xcode project, and change the scheme to `hydraplayer`.

Important

You're responsible for abiding by the terms of the license(s) associated with the code from the USD repo.

See Also

Render workflows

{} Using Metal to draw a view’s contents

Create a MetalKit view and a render pass to draw the view’s contents.

{} Drawing a triangle with Metal 4

Render a colorful, rotating 2D triangle by running draw commands with a render pipeline on a GPU.

{} Selecting device objects for graphics rendering

Switch dynamically between multiple GPUs to efficiently render to a display.

{} Customizing render pass setup

Render into an offscreen texture by creating a custom render pass.

{} Creating a custom Metal view

Implement a lightweight view for Metal rendering that’s customized to your app’s needs.

{} Calculating primitive visibility using depth testing

Determine which pixels are visible in a scene by using a depth texture.

{} Encoding indirect command buffers on the CPU

Reduce CPU overhead and simplify your command execution by reusing commands.

{} Implementing order-independent transparency with image blocks

Draw overlapping, transparent surfaces in any order by using tile shaders and image blocks.

{} Loading textures and models using Metal fast resource loading

Stream texture and buffer data directly from disk into Metal resources using fast resource loading.

{} Adjusting the level of detail using Metal mesh shaders

Choose and render meshes with several levels of detail using object and mesh shaders.

{} Culling occluded geometry using the visibility result buffer

Draw a scene without rendering hidden geometry by checking whether each object in the scene is visible.

{} Improving edge-rendering quality with multisample antialiasing (MSAA)

Apply MSAA to enhance the rendering of edges with custom resolve options and immediate and tile-based resolve paths.

{} Achieving smooth frame rates with a Metal display link

Pace rendering with minimal input latency while providing essential information to the operating system for power-efficient rendering, thermal mitigation, and the scheduling of sustainable workloads.