AdaptiveTessellationCS40

<https://github.com/walbourn/directx-sdk-samples>

This is the DirectX SDK's Direct3D 11 sample updated to use the Windows 10 SDK without any dependencies on legacy DirectX SDK content. This sample is a Win32 desktop DirectX 11.0 application for Windows 10, Windows 8.1, Windows 8, and Windows 7.

**This is based on the legacy DirectX SDK (June 2010) Win32 desktop sample. This is not intended for use with Windows Store apps, Windows RT, or universal Windows apps.**

# Description



AdaptiveTessellationCS40 demonstrates some adaptive tessellation techniques implemented using Compute Shader 4.0. You can use the radio buttons in the user interface to switch which tessellation schemes to use and observe how the tessellation pattern changes. The tessellation schemes implemented here are now identical to the triangle patch-based tessellation of Direct3D11 Tessellator Stage. So potentially the technique here could be a fallback solution for future games or other applications that use DirectX 11 hardware tessellation when they are running on devices that support only DirectX 10.x, because Compute Shader 4.0 runs on most DirectX 10.x-capable hardware.

Compute Shader allows more general algorithms to be implemented on the GPU. While the fully featured Compute Shader 5.0 requires Direct3D 11 hardware, a subset of that—Compute Shader 4.x—runs on existing Direct3D 10-compatible hardware, if the driver supports it.

## Overview

AdaptiveTessellationCS40 demonstrates how to use Compute Shader to manipulate geometry—the vertex data and index data of the tessellated mesh is generated on the fly in each frame by Compute Shaders, and later bound to the rendering pipeline for rendering. The following aspects of CS4.x are highlighted:

### Scattered writes to a structured buffer and a byte address buffer

A structured buffer can be read within all types of shaders and is good for data re-circulation between CS passes. A byte address buffer, also known as a raw buffer, can be bound as a vertex buffer and an index buffer, and thus, it is suitable to store geometry data that can be manipulated. Also note that, unlike CS 5.0, only one resource can be bound to a CS 4.x shader for output.

For more information about structured buffers and byte address buffers, see New Resource Types in the documentation for Direct3D 11.

### Use of shared memory

CS 4.x can read from any position of the shared memory, but it can only write to the position that is indexed by SV\_GroupIndex. Also, CS 4.x is limited to using a single shared memory. CS 5.0 doesn't have these limitations.

For more information about SV\_GroupIndex and the other system-value semantics that were introduced in DirectX 10, see Semantics in the documentation for HLSL.

### Use of ID3D11DeviceContext::Dispatch

The z dimension of the thread group grid is limited to 1 in CS 4.x; the upper limit for x and y dimensions are the same in CS 4.x and CS 5.0, which is 65536.

# Dependencies

DXUT-based samples typically make use of runtime HLSL compilation. Build-time compilation is recommended for all production Direct3D applications, but for experimentation and samples development runtime HLSL compilation is preferred. Therefore, the D3DCompile\*.DLL must be available in the search path when these programs are executed.

* When using the Windows 10 SDK and targeting Windows Vista or later, you can include the D3DCompile\_47 DLL side-by-side with your application copying the file from the REDIST folder.

%ProgramFiles(x86)%\Windows kits\10\Redist\D3D\ x86 or x64

# More Information

[Where is the DirectX SDK (2021 Edition)?](https://aka.ms/dxsdk)

[DXUT for Win32 Desktop Update](https://walbourn.github.io/dxut-for-win32-desktop-update/)

[Games for Windows and DirectX SDK blog](https://walbourn.github.io/)