Collision

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

This is the DirectX SDK's Direct3D 11 sample updated to use Visual Studio 2012 and the Windows SDK 8.0 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, Windows 7, and Windows Vista Service Pack 2 with the DirectX 11.0 runtime.

**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. A newer version of this sample for UWP and Xbox can be found here:** <https://github.com/Microsoft/Xbox-ATG-Samples>

# Description

This sample demonstrates [DirectXMath](http://go.microsoft.com/fwlink/?LinkID=615560) using the [DirectX Tool Kit](http://go.microsoft.com/fwlink/?LinkId=248929) and [DXUT for Direct3D 11](http://go.microsoft.com/fwlink/?LinkId=320437) framework for Win32 desktop applications.

## Collision

## https://code.msdn.microsoft.com/site/view/file/96797/1/Collision.jpg

## Create and Initialize Collision Object Structures

The DirectXMath Collision library has API functions to test collisions between many types of geometric shapes, including frustums, spheres, axis-aligned bounding boxes, oriented bounding boxes, triangles, planes, and rays. The Collision library header file also defines simple structures for most of these shapes. DirectXMath Math library types (XMVECTOR, and so on) are used for rays, triangles, and planes.

The collision objects are created in two groups—primary and secondary. Each collision test occurs between a primary object and a secondary object, and the collision result is recorded on the secondary object. Therefore, for secondary objects, a custom data structure is made for each type of collision object. The data structure contains a collision object structure and an integer flag to hold collision results.

The sample is visually organized so that each group of objects on the screen contains one primary object and a collection of secondary objects. Primary objects are made significantly larger than secondary objects.

Notice how the frustum object is created from a projection matrix. This way of constructing a frustum object makes it easy to create a frustum from your title's camera projection matrix, and then use the Collision library to do frustum testing on game objects.

## Animate Collision Objects

Note that only the secondary collision objects are animated. The sample uses simple mathematics functions to move these objects around and through the primary collision objects, providing opportunities for collisions to occur.

## Update the Camera

The sample uses the standard DXUT Model Viewer camera. You can select preset camera locations for the four sample collision groups through the UI or by pressing 1, 2, 3, or 4 on the keyboard.

## Test Collisions

The Collide function calls various functions in the Collision library on pairs of objects. The calls are grouped by primary object, so frustum tests come first, then axis-aligned box tests, and so on. Results from the collision tests, which are usually Boolean, are stored in the secondary object involved in each test.

For ray-object collision tests, the collision function returns a float parameter that represents the distance along the ray at which the collision occurred. If one of the ray-object collision tests succeeds, the float parameter is used to compute the 3D location of the collision. This location is stored as the center of a small cube, which is rendered later.

## Render Collision Objects

The Collision sample uses various drawing functions to render each collision object in wireframe graphics. The sample progresses through the following steps:

* It draws black gridlines to represent a ground plane below each object group. These gridlines are purely decorative, and help as a visual reference to establish the scene.
* It draws the primary collision objects in white.
* Using the results of each collision test, the sample draws the secondary objects in various colors. Red represents a collision; light green represents no collision. For the frustum collision tests, yellow represents a partial collision—that is, the object is partially inside the frustum, and partially outside.
* The sample renders ray-object collision tests in a separate group, and renders the collision ray in white. To better visually represent the extended direction of the collision ray, the sample renders a longer gray ray aligned with the collision ray.
* If one of the ray-object collision tests succeeds, the sample renders a small yellow cube at the collision location.

# 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 8.x SDK and targeting Windows Vista or later, you can include the D3DCompile\_46 or D3DCompile\_47 DLL side-by-side with your application copying the file from the REDIST folder.

%ProgramFiles(x86)%\Windows kits\8.0\Redist\D3D\arm, x86 or x64

%ProgramFiles(x86)%\Windows kits\8.1\Redist\D3D\arm, x86 or x64

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

# More Information

[Where is the DirectX SDK (2015 Edition)?](https://walbourn.github.io/where-is-the-directx-sdk-2015-edition/)

[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/)

[DirectX Tool Kit](http://go.microsoft.com/fwlink/?LinkId=248929)