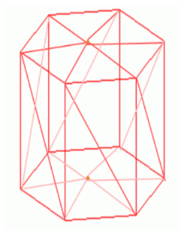
## Objectives

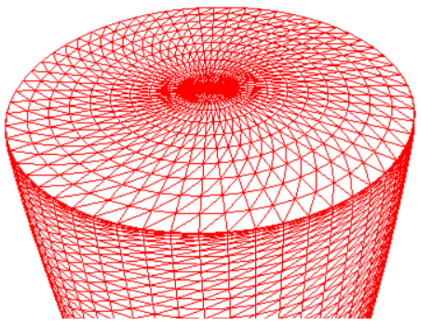
1. To discover how to procedurally generate and draw common geometric shapes like grids, cylinders, and spheres
2. To learn how to loading graphics assets using the AssImp library

## Exercise 1

Modify the vertex list of the cube to draw a hexagonal cylinder.

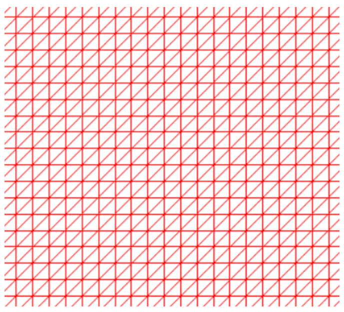


**(Optional**) Once you have achieved drawing of a hexagonal cylinder, you can further extend your work to draw a cylinder. (Hint: think the object as a collection of triangle strips with vertices defined by a list of circle radii $R_1$, $R_2$, …, $R_n$. then you can sample along the circle a sequence of points using the following formula: $x = R_i * \cos(\theta)\,, y = R_i * \sin(\theta)$ with a list of $\theta$ values from [0, $2\pi$].



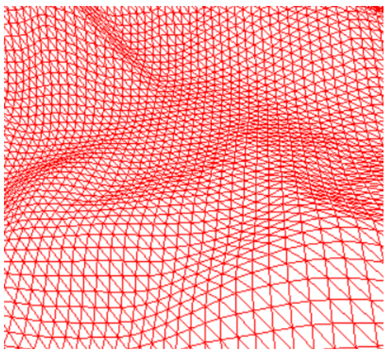
## Exercise 2

Modify the cube vertex list in the sample to specify a flat 3D grid and display it as a wireframe.



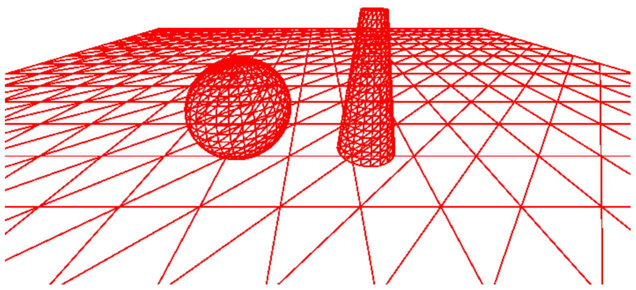
## Exercise 3

Specify different heights at different grid points for the 3D grid you created in Exercise 6 to create a terrain triangle mesh.



## Exercise 3

Have a look at the source code "*GeometryGenerator.cpp*" and "*GeometryGenerator.h*" as well as "*ShapesApp.cpp* " provided in the book of *Introduction to 3D Game Programming with Direct3D 12.0* at Github: https://github.com/d3dcoder/d3d12book/tree/master/Common, and https://github.com/d3dcoder/d3d12book/blob/master/Chapter%207%20Drawing%20in%20Direct3D%20Part%20II/Shapes/ShapesApp.cpp, and use the shapes defined in these files to draw the following three shapes.

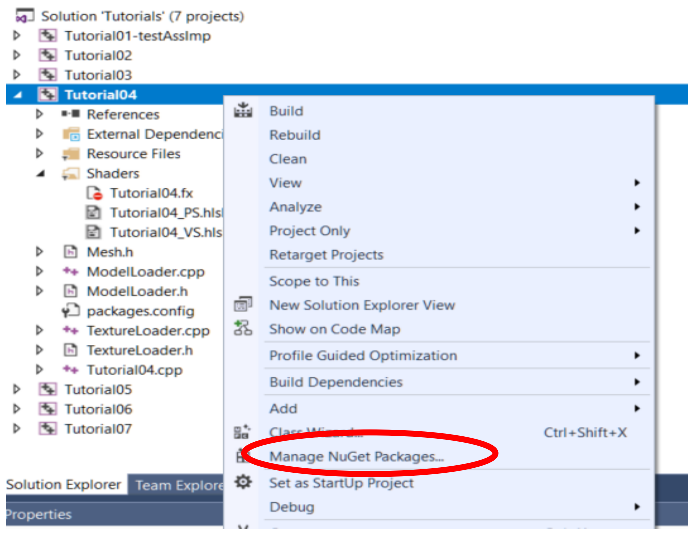


## Exercise 4. Loading graphics assets using AssImp

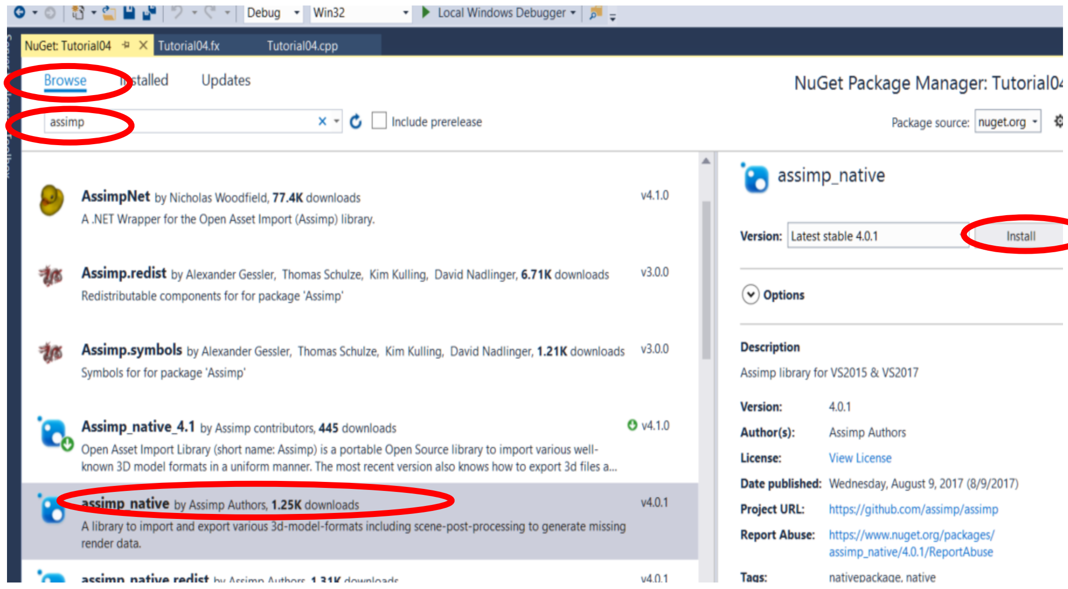
AssImp is a portable open source graphics asset importing library for loading commonly used 3D models. Please visit http://www.assimp.org/ for details of the asset importer.

AssImp library can be included in your project in a number of different ways. One simple way to use AssImp library in your project is to install AssImp by using the NuGet Packages Manager provided in VS 2017.

1. Right click the Solution Explorer or a specific project in the solution, choose Manage NuGet Packages. The



2. Choose Browse and search "Assimp", a list of Assimp libraries of different versions will be displayed. Select Assimp_native_4.1 if you want to use the latest version of the importer, or Assimp_native if you want to use the latest stable version.



**AssimpNet** by Nicholas Woodfield, **77.4K** downloads v4.1.0
A .NET Wrapper for the Open Asset Import (Assimp) library.

**Assimp.redist** by Alexander Gessler, Thomas Schulze, Kim Kulling, David Nadlinger, **6.71K** downloads v3.0.0
Redistributable components for for package 'Assimp'

**Assimp.symbols** by Alexander Gessler, Thomas Schulze, Kim Kulling, David Nadlinger, **1.21K** downloads v3.0.0
Symbols for for package 'Assimp'

**Assimp_native_4.1** by Assimp contributors, **445** downloads v4.1.0
Open Asset Import Library (short name: Assimp) is a portable Open Source library to import various well-known 3D model formats in a uniform manner. The most recent version also knows how to export 3d files a...

**assimp_native** by Assimp Authors, **1.25K** downloads v4.0.1
A library to import and export various 3d-model-formats including scene-post-processing to generate missing render data.

**assimp_native.redist** by Assimp Authors, 1.31K downloads v4.0.1

assimp_native

Version: Latest stable 4.0.1

Description
Assimp library for VS2015 & VS2017

Version: 4.0.1
Author(s): Assimp Authors
License: View License
Date published: Wednesday, August 9, 2017 (8/9/2017)
Project URL: https://github.com/assimp/assimp
Report Abuse: https://www.nuget.org/packages/assimp_native/4.0.1/ReportAbuse
Tags: nativepackage, native

3. Click the **Install** button to install the library.

4. Once the library is installed, you can then use it by add required Assimp header files in your program:

```
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
```

5. Load a model by creating an Assimp importer, say

```
Assimp::Importer importer;
const aiScene* scene = importer.ReadFile("..\\models\\teapot.obj",
aiProcess_Triangulate );
```

6. Create an Assimp mesh object for your model. For instance,

```
aiMesh* teapotMesh= scene->mMeshes[0];
```

7. Create an array of vertices from the loaded model using C++ vector template:

```
std::vector<SimpleVertex>mesh_vertices;
for (UINT i = 0; i < teapotMesh->mNumVertices; i++)
{
       SimpleVertex vertex;

       vertex.Pos.x = teapotMesh->mVertices[i].x;
       vertex.Pos.y = teapotMesh->mVertices[i].y;
       vertex.Pos.z = teapotMesh->mVertices[i].z;

       … …
       … …

       mesh_vertices.push_back(vertex);
}
```

8. Create an array of indices from the loaded model using C++ vector template:

```
std::vector<WORD>mesh_indices;  //need to include <vector> header
for (UINT i = 0; i < teapotMesh->mNumFaces; i++)
{
       aiFace face = teapotMesh->mFaces[i];

       for (UINT j = 0; j < face.mNumIndices; j++)
              mesh_indices.push_back(face.mIndices[j]);
}
```

9. Create using vertex buffer by replacing

```
bd.ByteWidth = sizeof( SimpleVertex ) * mesh_vertices.size();

InitData.pSysMem = mesh_vertices.data();
```

10. Do the same for creating the index buffer.