Real Time Graphics Lab B.

Week 2 – Lab B

Exercise 1.

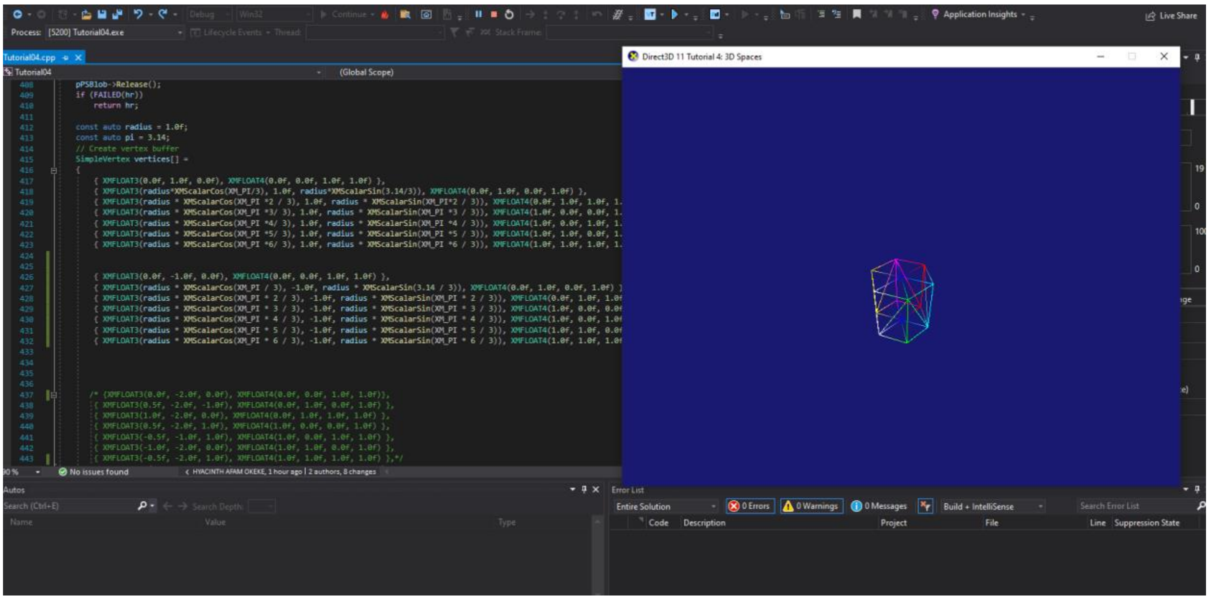
Modify the vertex list of the cube to draw a hexagonal cylinder..

Solution:

To create the vertex and indices of a cylinder

```
 const auto radius = 1.0f;
  const auto pi = 3.14;

 SimpleVertex vertices[] =


  {



    { XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius*XMScalarCos(XM_PI/3), 1.0f, radius*XMScalarSin(3.14/3)),
XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *2 / 3), 1.0f, radius *
XMScalarSin(XM_PI*2 / 3)), XMFLOAT4(0.0f, 1.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *3/ 3), 1.0f, radius * XMScalarSin(XM_PI
*3 / 3)), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *4/ 3), 1.0f, radius * XMScalarSin(XM_PI
*4 / 3)), XMFLOAT4(1.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *5/ 3), 1.0f, radius * XMScalarSin(XM_PI
*5 / 3)), XMFLOAT4(1.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *6/ 3), 1.0f, radius * XMScalarSin(XM_PI
*6 / 3)), XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f) },


    { XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI / 3), -1.0f, radius * XMScalarSin(3.14 /
3)), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 2 / 3), -1.0f, radius *
XMScalarSin(XM_PI * 2 / 3)), XMFLOAT4(0.0f, 1.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 3 / 3), -1.0f, radius *
XMScalarSin(XM_PI * 3 / 3)), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 4 / 3), -1.0f, radius *
XMScalarSin(XM_PI * 4 / 3)), XMFLOAT4(1.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 5 / 3), -1.0f, radius *
XMScalarSin(XM_PI * 5 / 3)), XMFLOAT4(1.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 6 / 3), -1.0f, radius *
XMScalarSin(XM_PI * 6 / 3)), XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f) },
```

Sample Output:



```
pPSBlob->Release();
if (FAILED(hr))
    return hr;

const auto radius = 1.0f;
const auto pi = 3.14;
// Create vertex buffer
SimpleVertex vertices[] =
{
    { XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius*XMScalarCos(XM_PI/3), 1.0f, radius*XMScalarSin(3.14/3)), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI *2 / 3), 1.0f, radius * XMScalarSin(XM_PI*2 / 3)), XMFLOAT4(0.0f, 1.0f, 1.0f, 1.
    { XMFLOAT3(radius * XMScalarCos(XM_PI *3/ 3), 1.0f, radius * XMScalarSin(XM_PI *3 / 3)), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.
    { XMFLOAT3(radius * XMScalarCos(XM_PI *4/ 3), 1.0f, radius * XMScalarSin(XM_PI *4 / 3)), XMFLOAT4(1.0f, 0.0f, 1.0f, 1.
    { XMFLOAT3(radius * XMScalarCos(XM_PI *5/ 3), 1.0f, radius * XMScalarSin(XM_PI *5 / 3)), XMFLOAT4(1.0f, 1.0f, 0.0f, 1.
    { XMFLOAT3(radius * XMScalarCos(XM_PI *6/ 3), 1.0f, radius * XMScalarSin(XM_PI *6 / 3)), XMFLOAT4(1.0f, 1.0f, 1.0f, 1.


    { XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(radius * XMScalarCos(XM_PI / 3), -1.0f, radius * XMScalarSin(3.14 / 3)), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) }
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 2 / 3), -1.0f, radius * XMScalarSin(XM_PI * 2 / 3)), XMFLOAT4(0.0f, 1.0f, 1.0f
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 3 / 3), -1.0f, radius * XMScalarSin(XM_PI * 3 / 3)), XMFLOAT4(1.0f, 0.0f, 0.0f
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 4 / 3), -1.0f, radius * XMScalarSin(XM_PI * 4 / 3)), XMFLOAT4(1.0f, 0.0f, 1.0f
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 5 / 3), -1.0f, radius * XMScalarSin(XM_PI * 5 / 3)), XMFLOAT4(1.0f, 1.0f, 0.0f
    { XMFLOAT3(radius * XMScalarCos(XM_PI * 6 / 3), -1.0f, radius * XMScalarSin(XM_PI * 6 / 3)), XMFLOAT4(1.0f, 1.0f, 1.0f




   /* {XMFLOAT3(0.0f, -2.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f)},
    { XMFLOAT3(0.5f, -2.0f, -1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(1.0f, -2.0f, 0.0f), XMFLOAT4(0.0f, 1.0f, 1.0f, 1.0f) },
    { XMFLOAT3(0.5f, -2.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) },
    { XMFLOAT3(-0.5f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 1.0f, 1.0f) },
    { XMFLOAT3(-1.0f, -2.0f, 0.0f), XMFLOAT4(1.0f, 1.0f, 0.0f, 1.0f) },
    { XMFLOAT3(-0.5f, -2.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f) },*/
```

Test data:

N/A

Reflection:

After several trials and failures I finally got the cylinder, in this exercise it was established that getting the vertices of the cylinder requires using a for loop or hard coding the inputs. In the case above I declared PI as a variable as well the x and y axis using Cos and Sin. These allowed generate the values for the vertices respectively. Furthermore I changed the values of the WORD indices for x and Y axis. I increased the number of triangles in the Drawindexed to correspond to the total number triangles required to complete the cylinder.

Metadata:

N/A

Further information:

N/A

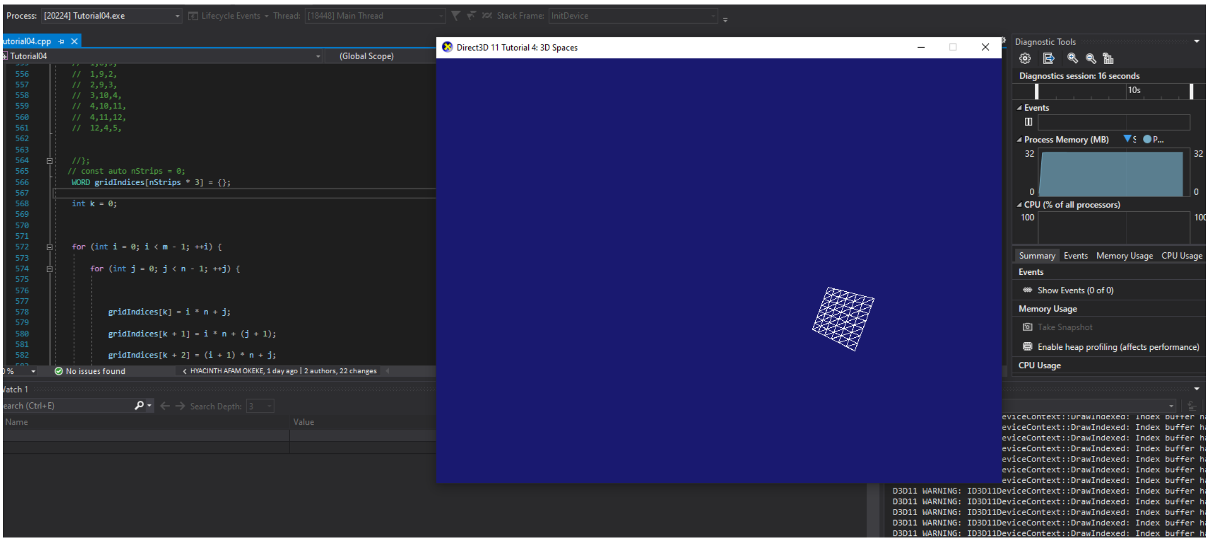
Exercise 2:

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

Solution:

```
const int m = 7;
   const  int n = 8;
    float w = -1.0f;
    float d = 1.0f;

    float halfWidth = 0.5f * w;

    float halfDepth = -1.5f * d;

    float dx = w / (n - 1);

    float dz = d / (m - 1);

    constexpr  auto nVertices = m * n;

    constexpr  auto nStrips = (m - 1) * (n - 1) * 2;

    SimpleVertex gridVertices[100] = {};

    for (int i = 0; i < m; ++i) {

        float z = halfDepth - i * dz;

        for (int j = 0; j < n; ++j) {

            float x = -halfWidth + j * dx;

            gridVertices[i * n + j].Pos = XMFLOAT3(x, 0.0f, z);


            gridVertices[i * n + j].Color = XMFLOAT4(1.0f, 1.0f, 1.0f, 0.0f);

        }

    };


WORD gridIndices[nStrips * 3] = {};

    int k = 0;


    for (int i = 0; i < m - 1; ++i) {

        for (int j = 0; j < n - 1; ++j) {
            gridIndices[k] = i * n + j;

            gridIndices[k + 1] = i * n + (j + 1);

            gridIndices[k + 2] = (i + 1) * n + j;

            gridIndices[k + 3] = (i + 1) * n + j;

            gridIndices[k + 4] = i * n + (j + 1);

            gridIndices[k + 5] = (i + 1) * n + (j + 1);


            k += 6;

        }

    };
```

Sample Output:



```
    //  1,9,2,
    //  2,9,3,
    //  3,10,4,
    //  4,10,11,
    //  4,11,12,
    //  12,4,5,


    //};
   // const auto nStrips = 0;
    WORD gridIndices[nStrips * 3] = {};

    int k = 0;


    for (int i = 0; i < m - 1; ++i) {

        for (int j = 0; j < n - 1; ++j) {


            gridIndices[k] = i * n + j;

            gridIndices[k + 1] = i * n + (j + 1);

            gridIndices[k + 2] = (i + 1) * n + j;
```

```
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
eviceContext::DrawIndexed: Index buffer ha
D3D11 WARNING: ID3D11DeviceContext::DrawIndexed: Index buffer ha
D3D11 WARNING: ID3D11DeviceContext::DrawIndexed: Index buffer ha
D3D11 WARNING: ID3D11DeviceContext::DrawIndexed: Index buffer ha
D3D11 WARNING: ID3D11DeviceContext::DrawIndexed: Index buffer ha
D3D11 WARNING: ID3D11DeviceContext::DrawIndexed: Index buffer ha
```

Reflection:

Generating the grid above, I used a for loop to iterate through generating all the triangles that made up the grid, further more I automated the Drawindexed value as thus" g_pImmediateContext->DrawIndexed(nStrips * 3, 0, 0);" this piece of code allowed for the display of the corresponding triangles as defined by the nStrips variable.