Realtime Graphics Lab Book

# Week 3 – Lab C

Date: 24th Oct 2024

## Exercise 1

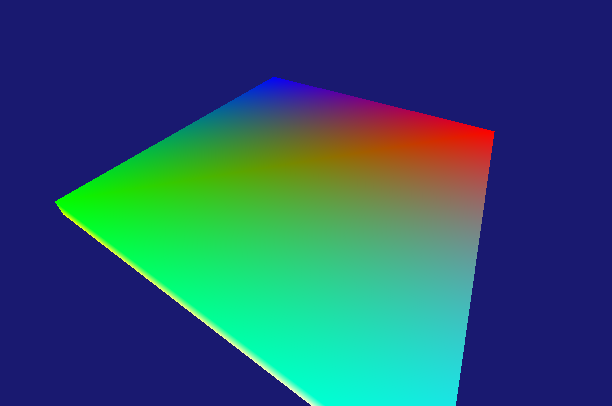
### Question: A screenshot of a computer program Description automatically generated

### Solution:

1. g\_World = XMMatrixScaling(1.0f, 4.0f, 1.0f) \* XMMatrixRotationY(t);
2. g\_World = XMMatrixScaling(4.0f, 0.2f, 4.0f) \* XMMatrixRotationY(t);
3. g\_World = XMMatrixScaling(4.0f, 1.0f, 1.0f) \* XMMatrixRotationY(t);

### Sample output:

A rainbow colored rectangle on a blue background

Description automatically generatedA colorful rectangle on a blue background

Description automatically generated

## Exercise 2:

A diagram of a model

Description automatically generated with medium confidence

### Solution:

In Init

auto box = gen.CreateBox(0.5f, 0.5f, 0.5f, 0);

RenderItem boxAxisIt;

boxAxisIt.Geo = &(box);

boxAxisIt.IndexCount = boxAxisIt.Geo->GetIndices16().size();

boxAxisIt.World = XMMatrixScaling(0.3f, 7.f, 0.3f) \* XMMatrixTranslation(-1.5f, 0.0f, 0.0f);

RenderItem boxAxisIt2;

boxAxisIt2.Geo = &(box);

boxAxisIt2.IndexCount = boxAxisIt.Geo->GetIndices16().size();

boxAxisIt2.World = XMMatrixScaling(0.3f, 7.f, 0.3f) \* XMMatrixTranslation(1.5f, 0.0f, 0.0f);

RenderItem box1It;

box1It.Geo = &(box);

box1It.IndexCount = box1It.Geo->GetIndices16().size();

RenderItem box2It;

box2It.Geo = &(box);

box2It.IndexCount = box1It.Geo->GetIndices16().size();

RenderItem gridIt;

auto grid = gen.CreateGrid(8, 8, 10, 10);

gridIt.Geo = &(grid);

gridIt.World = XMMatrixIdentity();

gridIt.World \*= XMMatrixTranslation(-0.0f, -1.0f, 0.0f);

gridIt.IndexCount = gridIt.Geo->GetIndices16().size();

g\_RenderItems.push\_back(gridIt);

g\_RenderItems.push\_back(boxAxisIt);

g\_RenderItems.push\_back(boxAxisIt2);

g\_RenderItems.push\_back(box1It);

g\_RenderItems.push\_back(box2It);

In Render:

### g\_RenderItems[3].World = XMMatrixTranslation(0.6f, 0.0f, 0.0f); // Orbit distance

### g\_RenderItems[3].World \*= XMMatrixRotationY(t); // Orbit rotation

### g\_RenderItems[3].World \*= XMMatrixTranslation(-1.5f, 0.0f, 0.0f); // Orbit pivot offset

### g\_RenderItems[4].World = XMMatrixTranslation(0.8f, 1.0f, 0.0f); // Orbit distance

### g\_RenderItems[4].World \*= XMMatrixRotationY(-t \* 2); // Orbit rotation

### g\_RenderItems[4].World \*= XMMatrixTranslation(1.5f, 0.0f, 0.0f); // Orbit pivot offset

### Sample output:

A colorful square object on a blue surface

Description automatically generated with medium confidence

A blue and green background with squares

Description automatically generatedA colorful gradient of a blue surface with a blue square and square objects

Description automatically generated with medium confidence

### Reflection:

*We have a box that serves as the visual of the orbit axis of the two boxes.*

*Then we created two boxes, to make them orbit around an axis:*

1. We translate them as the orbit distance from the wanted pivot
2. We do the accumulated rotation on each object
3. Then we translate them based on the pivot translation

## Exercise 3

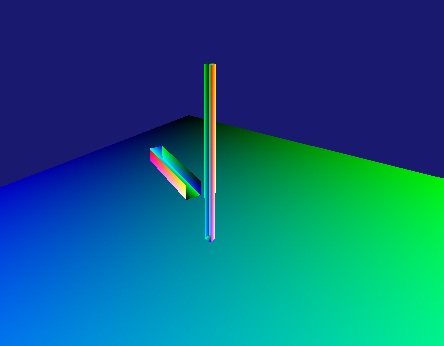
### A diagram of a circle with a circle and a circle with a circle and a circle with a circle with a circle and a circle with a circle with a circle with a circle with a circle with Description automatically generatedSolution:

### g\_RenderItems[2].World = XMMatrixTranslation(1.5f, 0.0f, 0.0f); // Orbit distance

### g\_RenderItems[2].World \*= XMMatrixScaling(0.5f, 0.5f, 3.0f); // Scale

### g\_RenderItems[2].World \*= XMMatrixRotationY(t); // Orbit rotation

### Sample output:



### Reflection:

*We can just easily scale up the z axis of the rotating cube AFTER OR BEFORE translating it.*

## Exercise 4

### Scale the cube into different sizes corresponding to the Sun, the Earth and the Moon respectively and then combine a set of rotation and translation transformations to animate a simple solar system.

### Solution:

RenderItem it;

auto box = gen.CreateSphere(1, 5, 5);

it.Geo = &(box);

it.IndexCount = it.Geo->GetIndices16().size();

g\_RenderItems.push\_back(it);

RenderItem it2;

auto spher = gen.CreateGrid(8, 8, 10, 10);

it2.Geo = &(spher);

it2.World = XMMatrixIdentity();

it2.World \*= XMMatrixTranslation(-0.0f, -1.0f, 0.0f);

it2.IndexCount = it2.Geo->GetIndices16().size();

g\_RenderItems.push\_back(it2);

RenderItem it3;

auto cyl = gen.CreateCylinder(1,0.3f,3, 8,8);

it3.Geo = &(cyl);

it3.World = XMMatrixIdentity();

it3.World \*= XMMatrixTranslation(-3.0f, -0.0f, 0.0f);

it3.IndexCount = it3.Geo->GetIndices16().size();

g\_RenderItems.push\_back(it3);

for (auto& tt : g\_RenderItems) {

D3D11\_BUFFER\_DESC bd = {};

bd.Usage = D3D11\_USAGE\_DEFAULT;

bd.ByteWidth = sizeof(GeometryGenerator::Vertex) \* tt.Geo->Vertices.size();

bd.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER;

bd.CPUAccessFlags = 0;

D3D11\_SUBRESOURCE\_DATA InitData = {};

InitData.pSysMem = tt.Geo->Vertices.data();

hr = g\_pd3dDevice->CreateBuffer(&bd, &InitData, &tt.verticesBuffer);

bd.Usage = D3D11\_USAGE\_DEFAULT;

bd.ByteWidth = sizeof(GeometryGenerator::uint16) \* tt.Geo->GetIndices16().size();

bd.BindFlags = D3D11\_BIND\_INDEX\_BUFFER;

bd.CPUAccessFlags = 0;

InitData.pSysMem = tt.Geo->GetIndices16().data();

hr = g\_pd3dDevice->CreateBuffer(&bd, &InitData, &tt.indicesBuffer);

}

### Sample Output:

A wireframe of a building

Description automatically generated with medium confidence

### Reflection:

*We can create the render item struct that would hold the vertices and indices buffer along with other necessary information, then we can iterate over every object to render it and set the correct corresponding buffers.*

## Exercise 5

### Draw two wireframe cubes. There are different ways of achieving this. One simple way is to pass a different world matrix to the vertex shader.

### Solution:

g\_World \*= XMMatrixTranslation(-2, 2, -1);

g\_World \*= XMMatrixScaling(0.5, 0.5, 0.5);

cb.mWorld = XMMatrixTranspose(g\_World);

g\_pImmediateContext->UpdateSubresource(g\_pConstantBuffer, 0, nullptr, &cb, 0, 0);

g\_pImmediateContext->DrawIndexed(36, 0, 0); // 36 vertices needed for 12 triangles in a triangle list

### Sample output:

A colorful cube on a blue background

Description automatically generated

### Reflection:

*We can simply use the same vertices and indices because the cube model didn’t change, so we change the world matrix to translate the cube again after drawing it the first time and scale it down a little bit. And then we draw it again so it looks like we have two cubes now on the screen by recycling the same vertices and indices.*

## Exercise 5

### Draw the cube as triangle strips by setting primitive topology as D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP.

### Solution:

WORD wallIndices[] =

{

// Front face

0, 1, 4, 5,

// Right face

1, 2, 5, 6,

// Back face

2, 3, 6, 7,

// Left face

3, 0, 7, 4

};

WORD topIndices[] =

{

3, 2, 0, 1

};

WORD bottomIndices[] =

{

4, 5, 7, 6

};

### Sample output:

A colorful cube on a blue background

Description automatically generated

### Reflection:

*We cannot render the whole cube using triangle strip in one go, we can separate the cube into three triangle strips instead, the walls of the cube are a triangle strip, the bottom and the top faces are two individual triangle strips.  
So then we must separate our indices into 3 different arrays, and then we will draw each individual array as a triangle strip*