Realtime Graphics Lab Book

# Week 3 – Lab C

Date: 24th Oct 2024

## Exercise 1

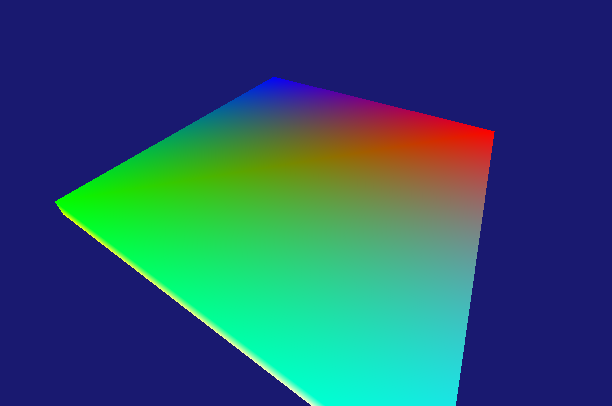
### Question: A screenshot of a computer program Description automatically generated

### 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

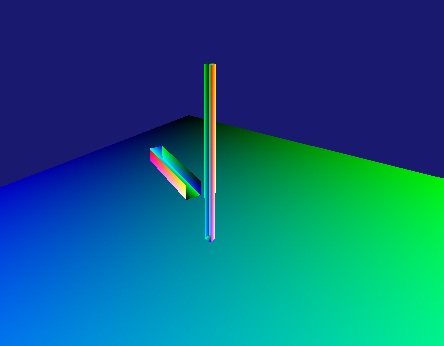
### A diagram of a circle with a circle and a circle with a circle and a circle with a circle with a circle and a circle with a circle with a circle with a circle with a circle with Description automatically generatedSolution:

### 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:

float moonRotationAroundEarthSpeed = t / 2;

float moonRotationAroundSelfSpeed = t \* 4;

float earthRotationAroundSunSpeed = t / 4;

float earthRotationAroundSelfSpeed = t / 2;

float sunRotationSpeed = t / 8;

g\_RenderItems[0].World = XMMatrixScaling(3.0f, 3.0f, 3.0f) \* XMMatrixRotationY(sunRotationSpeed); // Sun

g\_RenderItems[1].World = XMMatrixScaling(1.0f, 1.0f, 1.0f) \* XMMatrixRotationY(earthRotationAroundSelfSpeed) \*

XMMatrixTranslation(-6.0f, 0.0f, 0.0f) \* XMMatrixRotationY(earthRotationAroundSunSpeed); // Earth

g\_RenderItems[2].World = XMMatrixScaling(.5f, .5f, .5f) \* XMMatrixRotationY(moonRotationAroundSelfSpeed)

\* XMMatrixTranslation(-2.f, 0.0f, 0.0f) \* XMMatrixRotationY(moonRotationAroundEarthSpeed)

\* XMMatrixTranslation(-6.0f, 0.0f, 0.0f) \* XMMatrixRotationY(earthRotationAroundSunSpeed); // Moon

### Sample Output:

A colorful circles on a blue background

Description automatically generatedA group of colorful circles

Description automatically generated

### Reflection:

*We constructed three spheres to draw, each sphere has its own rotation around itself, the earth rotates around the biggest sphere and around itself, and the moon rotates around itself, around earth and around the sun.*

## Exercise 5

### In the Tutorial04, the view transformation and projection transformation are created from the XMMatrixLookAtLH( ) method and the XMMatrixPerspectiveFovLH() method. To develop a better understanding of the two transformations, you can directly define the matrices in your c++ program and observe if you can get exactly the same effect. You can define the two matrices directly using XMMatrixSet( ) method.

### Solution:

// Initialize the world matrix

g\_World = XMMatrixIdentity();

// Initialize the view matrix manually

XMVECTOR Eye = XMVectorSet(.0f, 15.5f, 0.0f, 0.0f);

XMVECTOR At = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f);

XMVECTOR Up = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f);

XMVECTOR zaxis = XMVector3Normalize(XMVectorSubtract(At, Eye));

XMVECTOR xaxis = XMVector3Normalize(XMVector3Cross(Up, zaxis));

XMVECTOR yaxis = XMVector3Cross(zaxis, xaxis);

g\_View = XMMATRIX(

xaxis.m128\_f32[0], yaxis.m128\_f32[0], zaxis.m128\_f32[0], 0.0f,

xaxis.m128\_f32[1], yaxis.m128\_f32[1], zaxis.m128\_f32[1], 0.0f,

xaxis.m128\_f32[2], yaxis.m128\_f32[2], zaxis.m128\_f32[2], 0.0f,

-XMVectorGetX(XMVector3Dot(xaxis, Eye)), -XMVectorGetX(XMVector3Dot(yaxis, Eye)), -XMVectorGetX(XMVector3Dot(zaxis, Eye)), 1.0f

);

// Initialize the projection matrix manually

float fov = XM\_PIDIV2;

float aspectRatio = width / (FLOAT)height;

float nearZ = 0.01f;

float farZ = 100.0f;

float yScale = 1.0f / tan(fov / 2);

float xScale = yScale / aspectRatio;

g\_Projection = XMMATRIX(

xScale, 0.0f, 0.0f, 0.0f,

0.0f, yScale, 0.0f, 0.0f,

0.0f, 0.0f, farZ / (farZ - nearZ), 1.0f,

0.0f, 0.0f, -nearZ \* farZ / (farZ - nearZ), 0.0f

);

### Sample output:

A colorful circles on a blue background

Description automatically generated

### Reflection:

*Instead of calculating the view and project matrices using methods, we can calculate them manually by understanding the relying math behind it.*

The view matrix is constructed by defining the camera's position (Eye), the point it looks at (At), and the up direction (Up). It then calculates the basis vectors (xaxis, yaxis, zaxis) and constructs the view matrix using these vectors and the camera position.

The projection matrix is calculated using the field of view (FOV), aspect ratio, and near and far clipping planes. It computes scale factors for the x and y axes and constructs the projection matrix to transform view coordinates into normalized device coordinates (NDC).