Lab Books for 700106 & 700120

Contents

Real Time Graphics (700106) Labs

Week 1 – Lab 1

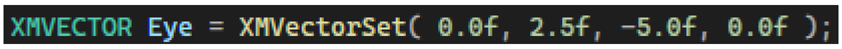
Date: 28th Sep 2022

Q1. Exercise 0

Question:

Change the “Eye” view so that you can see the top of the animated cube

Solution:

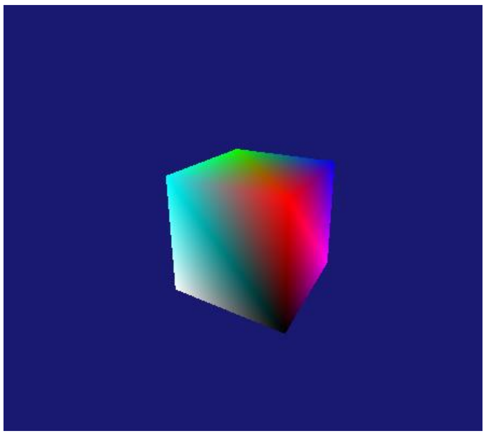


```
XMVECTOR Eye = XMVectorSet( 0.0f, 2.5f, -5.0f, 0.0f );
```

Test data:

n/a

Sample output:



Reflection:

This was very straight-forward just had to change the y position of the “Eye” XMVECTOR

Metadata:

Change of view

Further information:

n/a

Q2. Exercise 1

Question:

Modify the vertex list in indices[] or modify the parameters in the DrawIndexed( ) to draw:

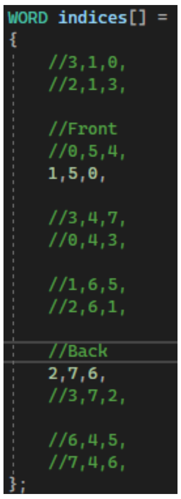
1. two triangles

2. one face of the cube

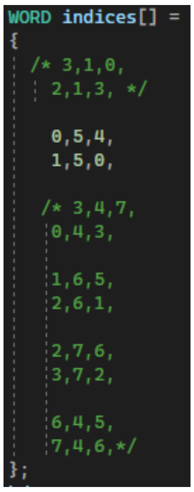
3. the four walls of the cube

Solution:

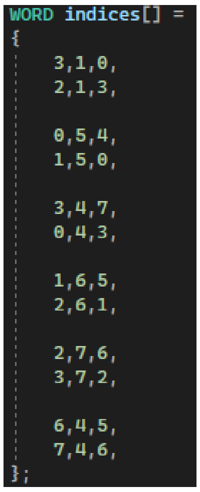
(1) (2) (3)



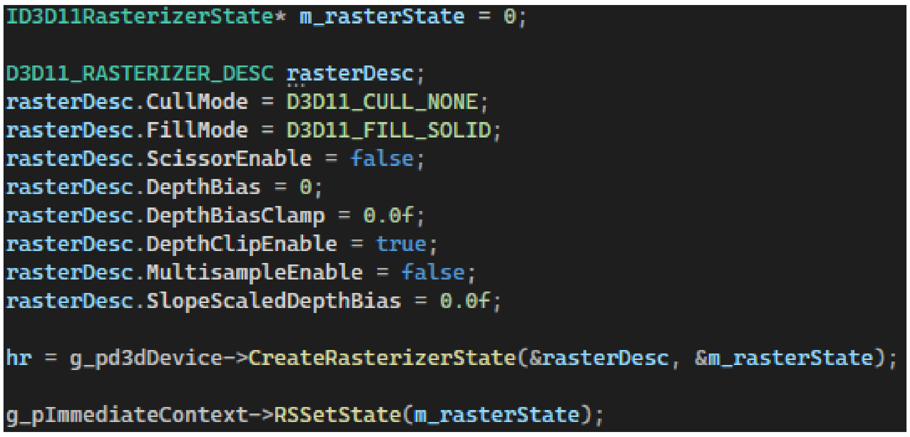
```
WORD indices[] =
{
    //3,1,0,
    //2,1,3,

    //Front
    //0,5,4,
    1,5,0,

    //3,4,7,
    //0,4,3,

    //1,6,5,
    //2,6,1,

    //Back
    2,7,6,
    //3,7,2,

    //6,4,5,
    //7,4,6,
};
```



```
WORD indices[] =
{
  /* 3,1,0,
    2,1,3, */

    0,5,4,
    1,5,0,

   /* 3,4,7,
    0,4,3,

    1,6,5,
    2,6,1,

    2,7,6,
    3,7,2,

    6,4,5,
    7,4,6,*/
};
```



```
WORD indices[] =
{
    3,1,0,
    2,1,3,

    0,5,4,
    1,5,0,

    3,4,7,
    0,4,3,

    1,6,5,
    2,6,1,

    2,7,6,
    3,7,2,

    6,4,5,
    7,4,6,
};
```



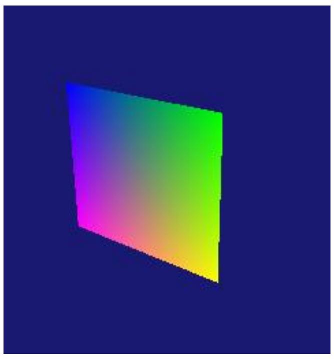
```
ID3D11RasterizerState* m_rasterState = 0;

D3D11_RASTERIZER_DESC rasterDesc;
rasterDesc.CullMode = D3D11_CULL_NONE;
rasterDesc.FillMode = D3D11_FILL_SOLID;
rasterDesc.ScissorEnable = false;
rasterDesc.DepthBias = 0;
rasterDesc.DepthBiasClamp = 0.0f;
rasterDesc.DepthClipEnable = true;
rasterDesc.MultisampleEnable = false;
rasterDesc.SlopeScaledDepthBias = 0.0f;

hr = g_pd3dDevice->CreateRasterizerState(&rasterDesc, &m_rasterState);

g_pImmediateContext->RSSetState(m_rasterState);
```

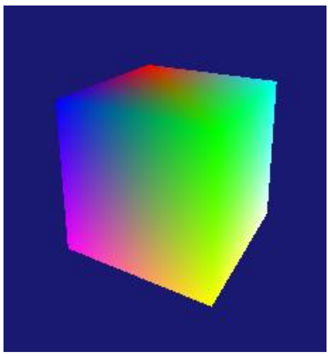
Test data:

n/a

Sample output:

1. 2. 3.





Reflection:

The rasterizer seems complicated at the moment but everything else was fairly easy

Metadata:

Faces

Further information:

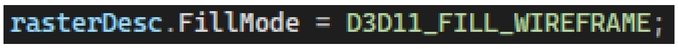
n/a

Q3. Exercise 2

Question:

Draw the cube as a wireframe

Solution:

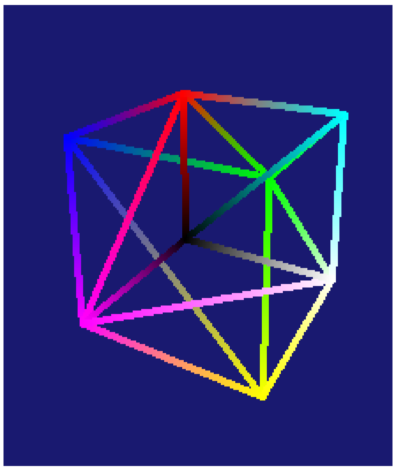


```
rasterDesc.FillMode = D3D11_FILL_WIREFRAME;
```

Test data:

n/a

Sample output:



Reflection:

This was really easy to do. I only had to change half a line of code

Metadata:

Wireframe

Further information:

n/a

Q4. Exercise 3

Question:

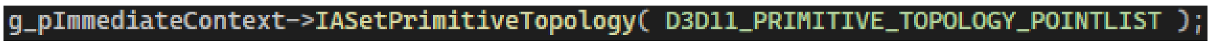
Modify the parameter in “IASetPrimitiveTopology( )” and “indices[]” to draw:

1. A list of points corresponding to the cube’s eight vertices.

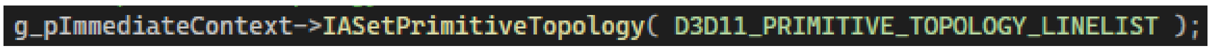
2. The 12 edges of the cube (not as a wireframe triangle mesh).

Solution:

1.



```
g_pImmediateContext->IASetPrimitiveTopology( D3D11_PRIMITIVE_TOPOLOGY_POINTLIST );
```



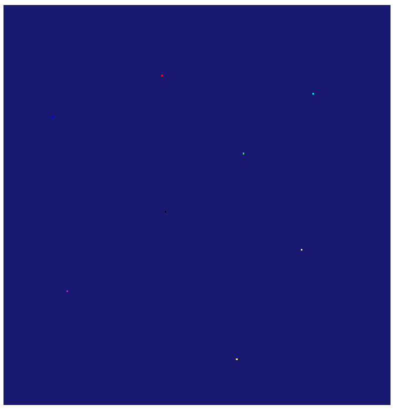
```
g_pImmediateContext->IASetPrimitiveTopology( D3D11_PRIMITIVE_TOPOLOGY_LINELIST );
```

Test data:

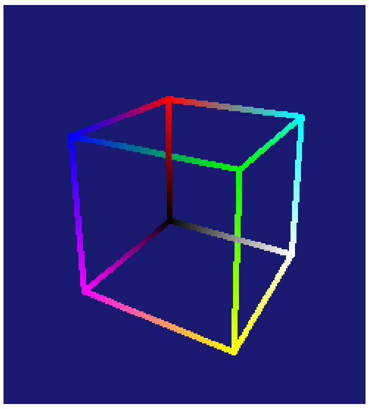
n/a

Sample output:

1.



2.



Reflection:

It took a while to figure out part 2, I was drawing too many lines for what was needed

Metadata:

Topology

Further information:

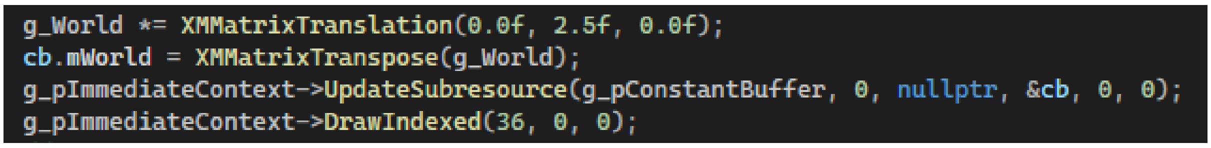
n/a

Q5. Exercise 4

Question:

Draw two wireframe cubes

Solution:

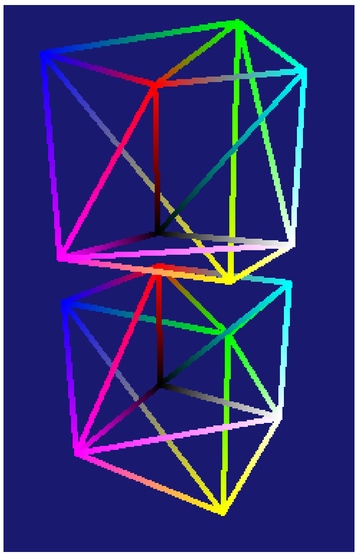


```
g_World *= XMMatrixTranslation(0.0f, 2.5f, 0.0f);
cb.mWorld = XMMatrixTranspose(g_World);
g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb, 0, 0);
g_pImmediateContext->DrawIndexed(36, 0, 0);
```

Test data:

n/a

Sample output:



Reflection:

Figuring out where I needed to put the code was the only hard part, but at least now I know.

Metadata:

Twice the cubes

Further information:

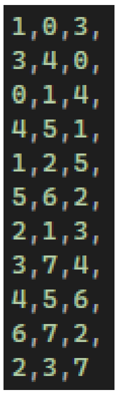
n/a

Q6. Exercise 5

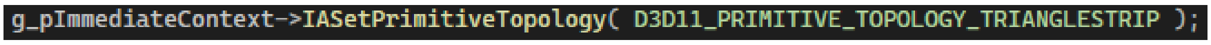
Question:

Draw the cube as triangle strips by setting primitive topology as D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP

Solution:



```
1,0,3,
3,4,0,
0,1,4,
4,5,1,
1,2,5,
5,6,2,
2,1,3,
3,7,4,
4,5,6,
6,7,2,
2,3,7
```

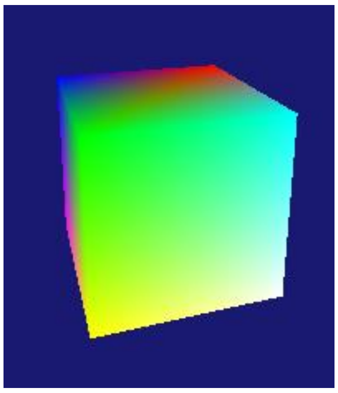


```
g_pImmediateContext->IASetPrimitiveTopology( D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP );
```

Test data:

n/a

Sample output:



Reflection:

This took me way too long and I still couldn't get it perfect. I understand triangle-strip conceptually, I just can't visualise it clearly enough in my head to implement it perfectly.

Metadata:

Triangle strip

Further information:

How can I better visualise it to implement it correctly?

Week 2 – Lab 2

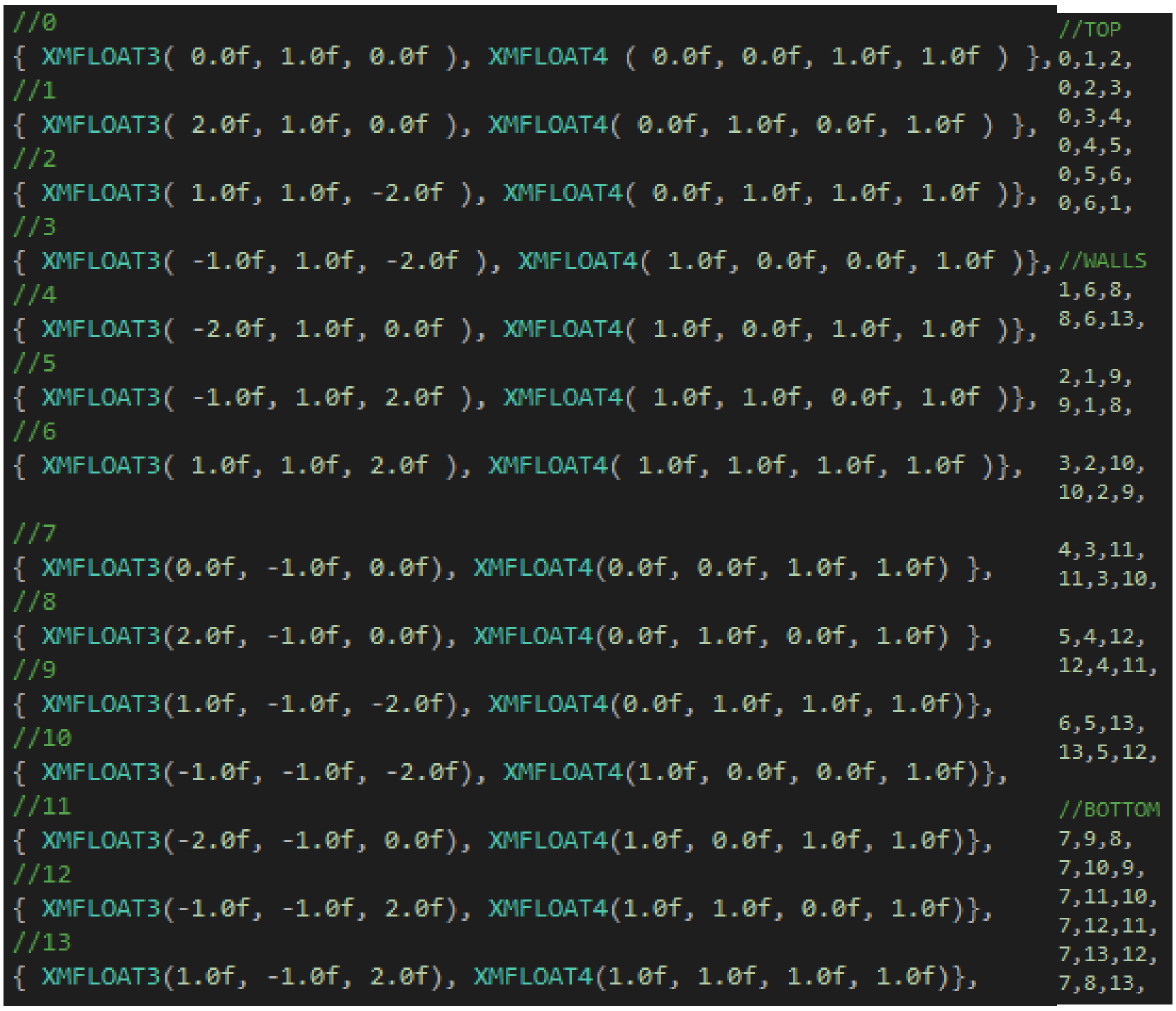
Date: 05/10/2022

Q1. Exercise 1

Question:

Modify the vertex list and indices to create a hexagonal cylinder

Solution:



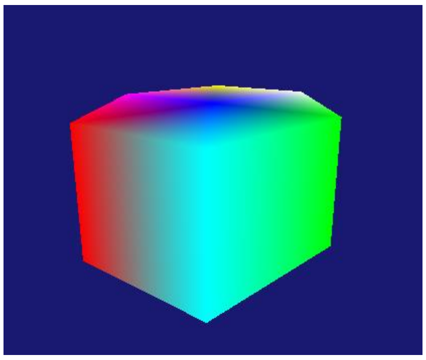
```
//0
{ XMFLOAT3( 0.0f, 1.0f, 0.0f ), XMFLOAT4 ( 0.0f, 0.0f, 1.0f, 1.0f ) },
//1
{ XMFLOAT3( 2.0f, 1.0f, 0.0f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
//2
{ XMFLOAT3( 1.0f, 1.0f, -2.0f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f )},
//3
{ XMFLOAT3( -1.0f, 1.0f, -2.0f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f )},
//4
{ XMFLOAT3( -2.0f, 1.0f, 0.0f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f )},
//5
{ XMFLOAT3( -1.0f, 1.0f, 2.0f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f )},
//6
{ XMFLOAT3( 1.0f, 1.0f, 2.0f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f )},

//7
{ XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f) },
//8
{ XMFLOAT3(2.0f, -1.0f, 0.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f) },
//9
{ XMFLOAT3(1.0f, -1.0f, -2.0f), XMFLOAT4(0.0f, 1.0f, 1.0f, 1.0f)},
//10
{ XMFLOAT3(-1.0f, -1.0f, -2.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f)},
//11
{ XMFLOAT3(-2.0f, -1.0f, 0.0f), XMFLOAT4(1.0f, 0.0f, 1.0f, 1.0f)},
//12
{ XMFLOAT3(-1.0f, -1.0f, 2.0f), XMFLOAT4(1.0f, 1.0f, 0.0f, 1.0f)},
//13
{ XMFLOAT3(1.0f, -1.0f, 2.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f)},
```

```
//TOP
0,1,2,
0,2,3,
0,3,4,
0,4,5,
0,5,6,
0,6,1,
//WALLS
1,6,8,
8,6,13,

2,1,9,
9,1,8,

3,2,10,
10,2,9,

4,3,11,
11,3,10,

5,4,12,
12,4,11,

6,5,13,
13,5,12,

//BOTTOM
7,9,8,
7,10,9,
7,11,10,
7,12,11,
7,13,12,
7,8,13,
```

Test data:

n/a

Sample output:



Reflection:

This took way longer the it needed because I missed a single line. I forgot to update the size of the vertex list. `bd.ByteWidth = sizeof( SimpleVertex ) * 14;`

Metadata:

Hex

Further information:

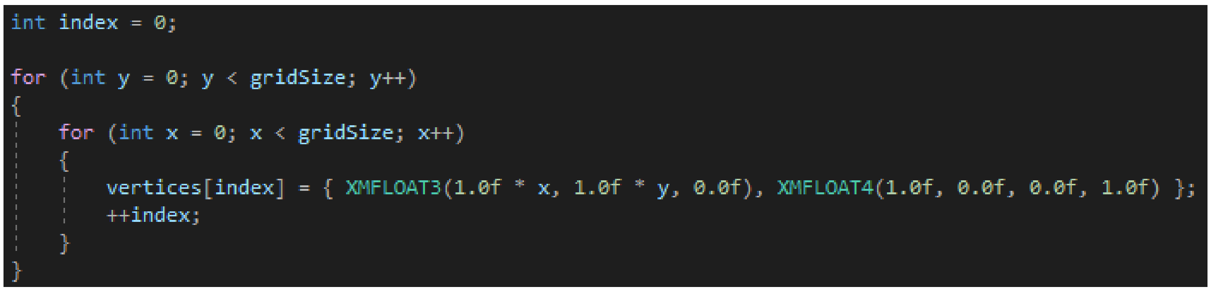
n/a

Q2. Exercise 2

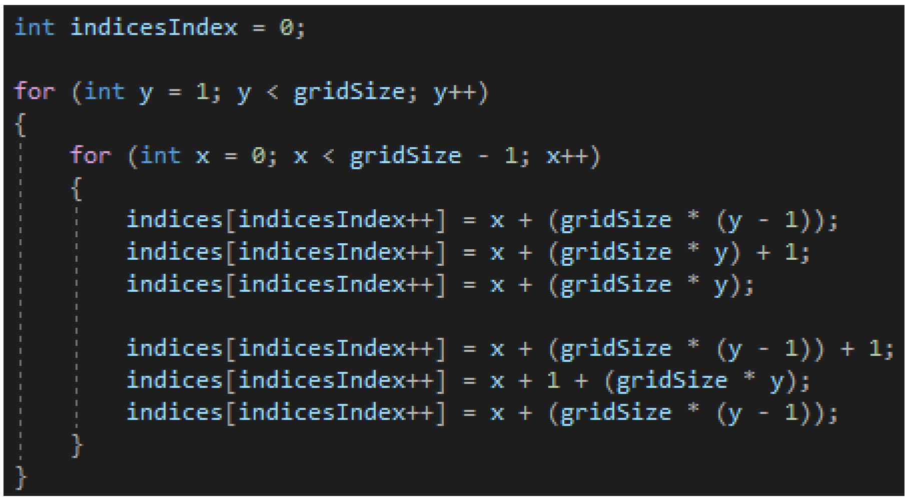
Question:

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

Solution:



```
int index = 0;

for (int y = 0; y < gridSize; y++)
{
    for (int x = 0; x < gridSize; x++)
    {
        vertices[index] = { XMFLOAT3(1.0f * x, 1.0f * y, 0.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) };
        ++index;
    }
}
```

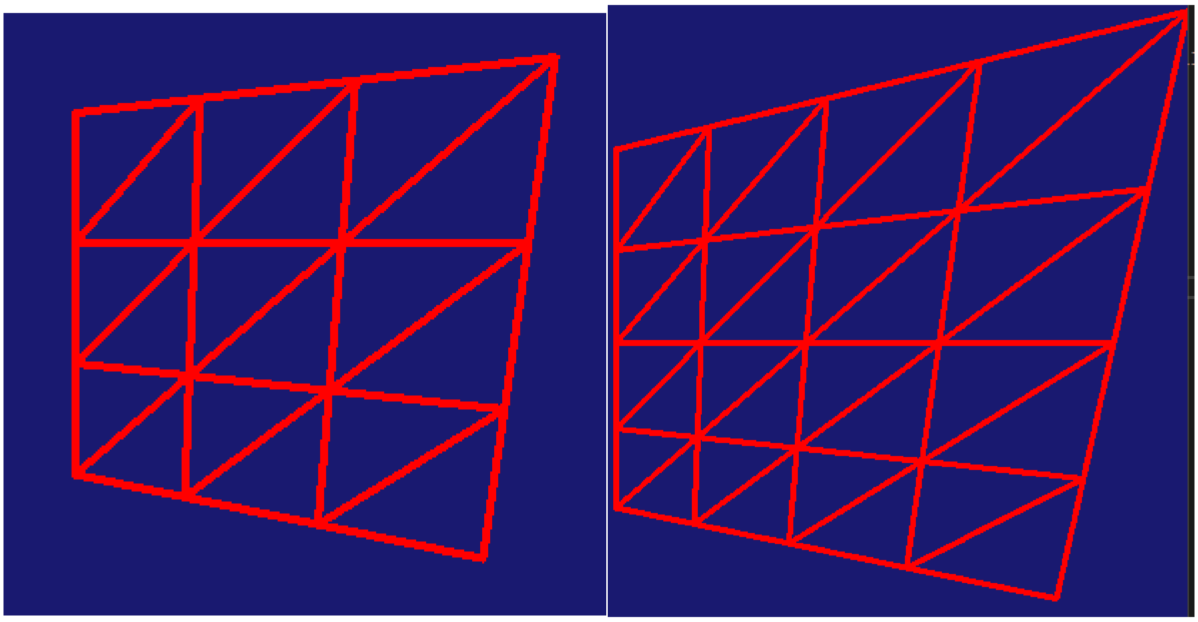


```
int indicesIndex = 0;

for (int y = 1; y < gridSize; y++)
{
    for (int x = 0; x < gridSize - 1; x++)
    {
        indices[indicesIndex++] = x + (gridSize * (y - 1));
        indices[indicesIndex++] = x + (gridSize * y) + 1;
        indices[indicesIndex++] = x + (gridSize * y);

        indices[indicesIndex++] = x + (gridSize * (y - 1)) + 1;
        indices[indicesIndex++] = x + 1 + (gridSize * y);
        indices[indicesIndex++] = x + (gridSize * (y - 1));
    }
}
```

Test data:

n/a

Sample output:



Reflection:

The procedural indices took forever to figure out how to do, I was over engineering the solution needed.

Metadata:

Procedural

Further information:

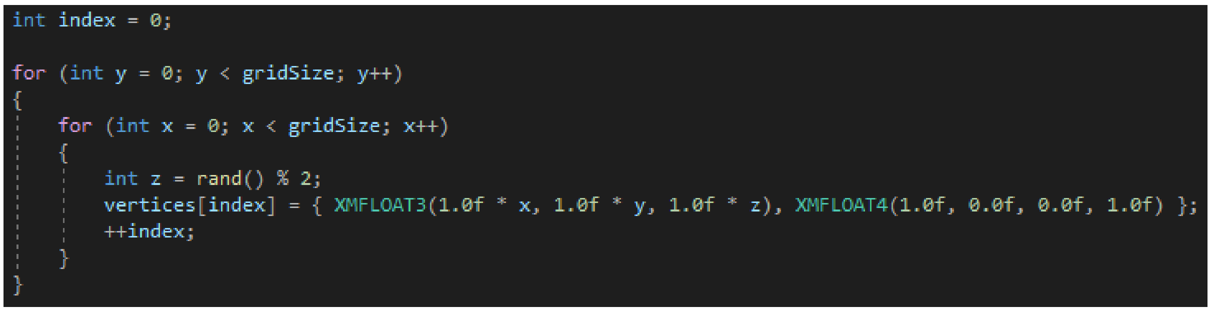
n/a

Q3. Exercise 3

Question:

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

Solution:

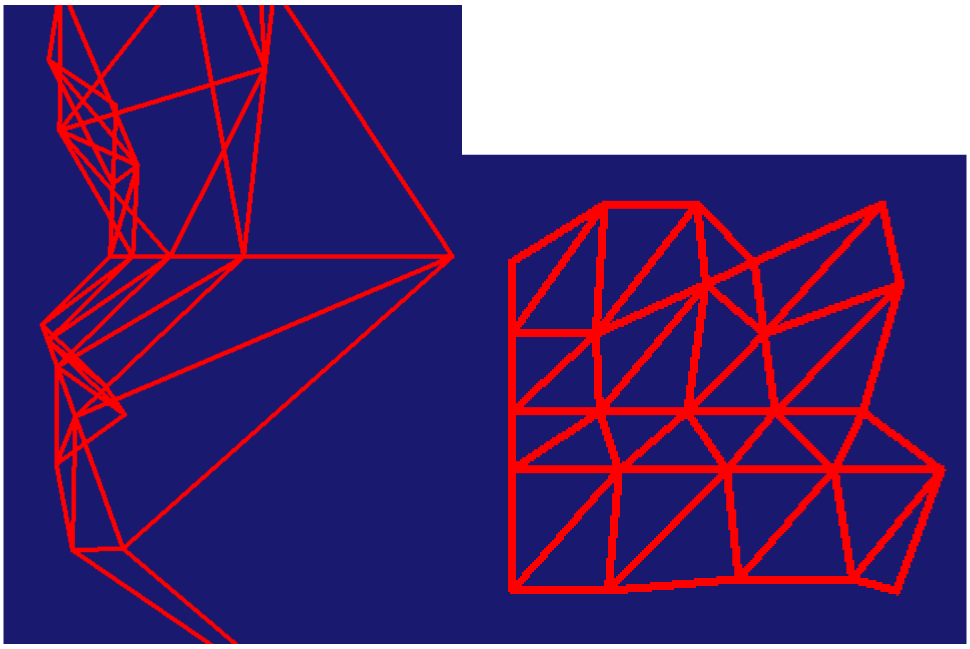


```
int index = 0;

for (int y = 0; y < gridSize; y++)
{
    for (int x = 0; x < gridSize; x++)
    {
        int z = rand() % 2;
        vertices[index] = { XMFLOAT3(1.0f * x, 1.0f * y, 1.0f * z), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) };
        ++index;
    }
}
```

Test data:

n/a

Sample output:



Reflection:

This was pretty straight-forward once I got the procedural indices to work. It's a bit hard to get a good angle to show off that it has got an z axis. I probably should have done a wider range because at the moment the z axis is only ever 0.0f or 1.0f.

Metadata:

Terrain

Further information:

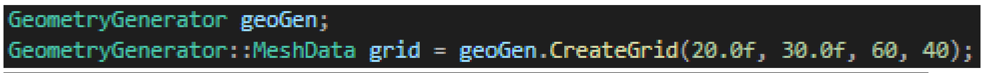
n/a

Q4. Exercise 4

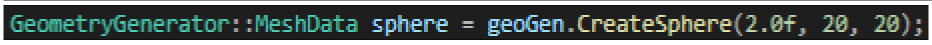
Question:

Use e “GeometryGenerator.cpp” , “GeometryGenerator.h” and “ShapesApp.cpp” to draw a sphere, cylinder and plane.

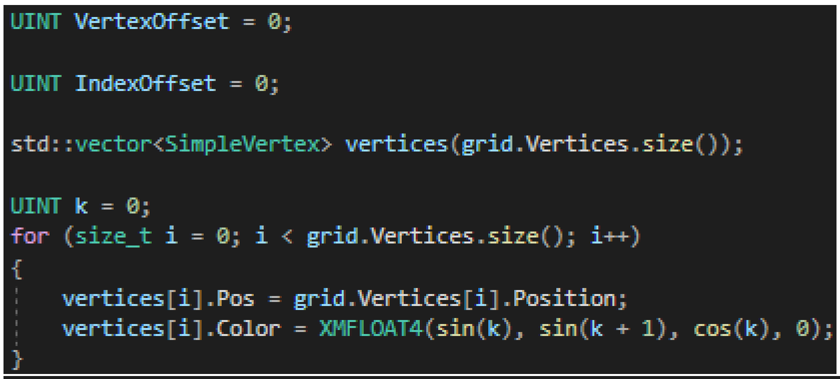
Solution:



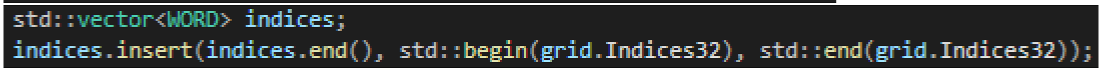
```
GeometryGenerator geoGen;
GeometryGenerator::MeshData grid = geoGen.CreateGrid(20.0f, 30.0f, 60, 40);
```



```
GeometryGenerator::MeshData sphere = geoGen.CreateSphere(2.0f, 20, 20);
```



```
UINT VertexOffset = 0;

UINT IndexOffset = 0;

std::vector<SimpleVertex> vertices(grid.Vertices.size());

UINT k = 0;
for (size_t i = 0; i < grid.Vertices.size(); i++)
{
    vertices[i].Pos = grid.Vertices[i].Position;
    vertices[i].Color = XMFLOAT4(sin(k), sin(k + 1), cos(k), 0);
}
```



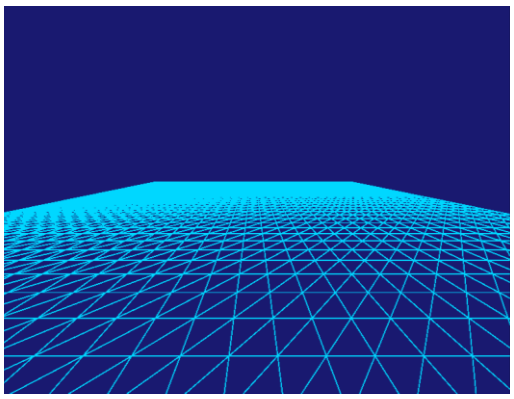
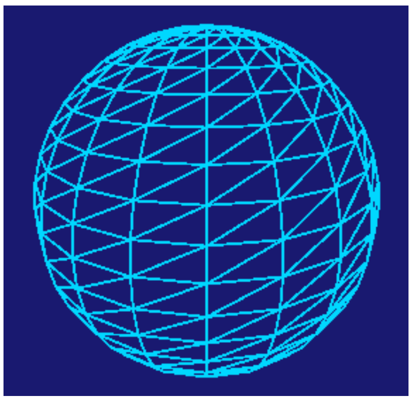
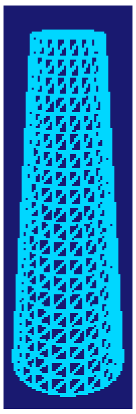
```
std::vector<WORD> indices;
indices.insert(indices.end(), std::begin(grid.Indices32), std::end(grid.Indices32));
```

These are pretty much the same for all the shapes the only difference is the keyword changes e.g. “grid”.

Test data:

n/a

Sample output:

Reflection:

This was pretty straight forward once I had spotted the simple mistake I had made. I’m noy sure how you would draw all of them as once like in the “ShapesApp.cpp” example.

Metadata:

Geometry generator

Further information:

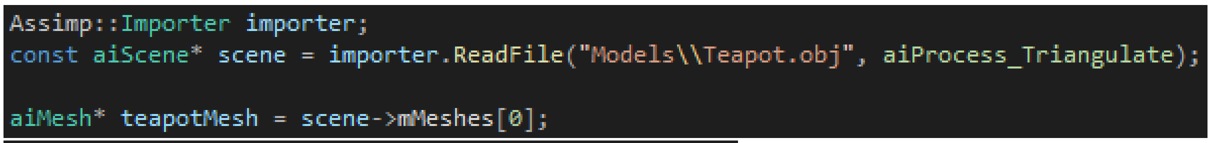
n/a

Q5. Exercise 5 Loading graphics assets using AssImp

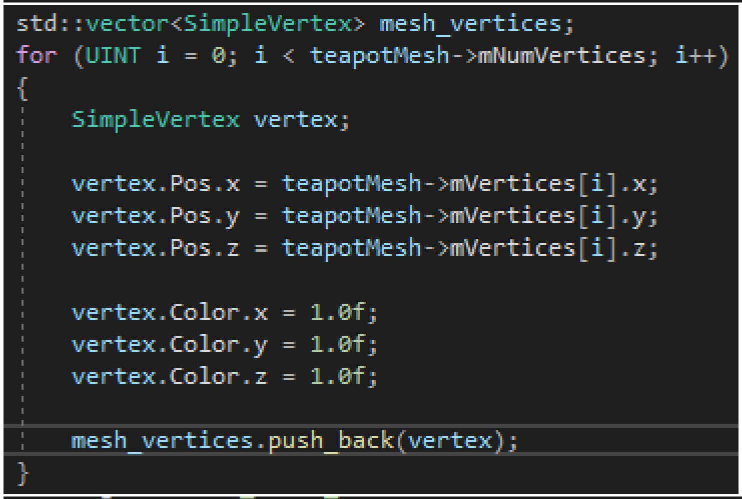
Question:

Use AssImp to draw the teapot model given

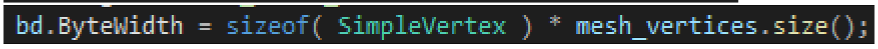
Solution:



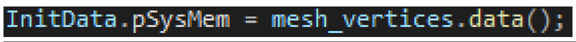
```
Assimp::Importer importer;
const aiScene* scene = importer.ReadFile("Models\\Teapot.obj", aiProcess_Triangulate);

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



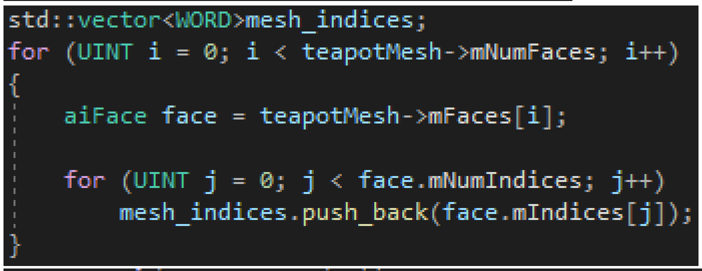
```
std::vector<SimpleVertex> mesh_vertices;
for (UINT i = 0; i < teapotMesh->mNumVertices; i++)
{
    SimpleVertex vertex;

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

    vertex.Color.x = 1.0f;
    vertex.Color.y = 1.0f;
    vertex.Color.z = 1.0f;

    mesh_vertices.push_back(vertex);
}
```



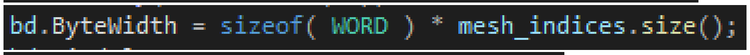
```
bd.ByteWidth = sizeof( SimpleVertex ) * mesh_vertices.size();
```



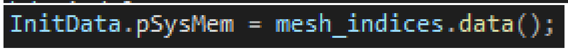
```
InitData.pSysMem = mesh_vertices.data();
```



```
std::vector<WORD>mesh_indices;
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]);
}
```



```
bd.ByteWidth = sizeof( WORD ) * mesh_indices.size();
```

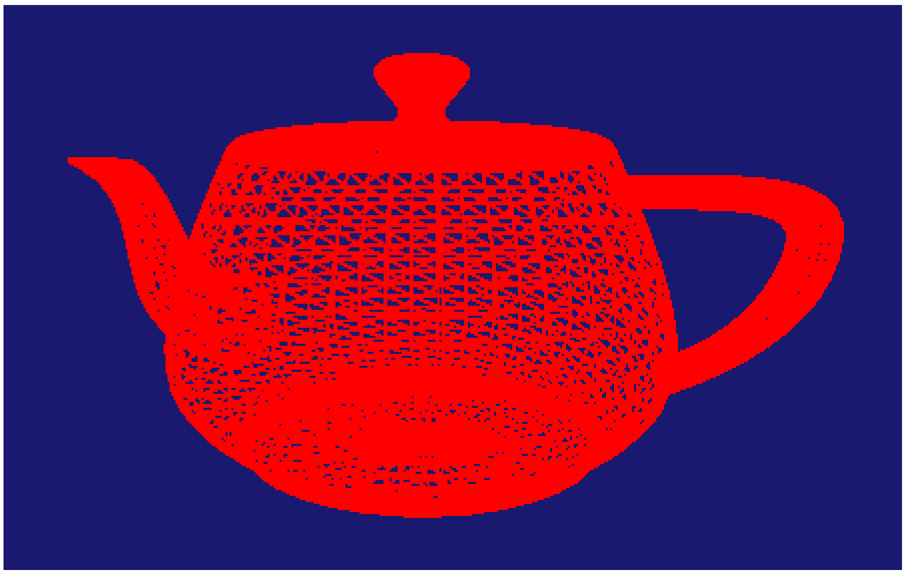


```
InitData.pSysMem = mesh_indices.data();
```

Test data:

n/a

Sample output:



Reflection:

Finally got this fixed nearly 10 weeks later, once the error with the includes and min had been fixed, I was having errors with the obj file becoming corrupt. I fixed this by loading the given obj into blender and then exporting it as a new obj which seemed to fix the problem but it might be because I tried doing this in the final lab code.

Metadata:

Assimp

Further information:

n/a

Week 3 – Lab 3

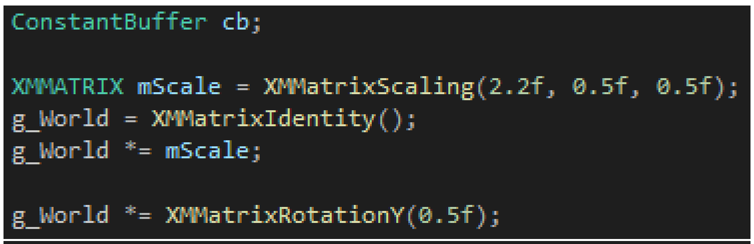
Date: 11/10/2022

Q1. Exercise 1

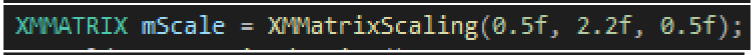
Question:

Transform the cube using scaling transformation

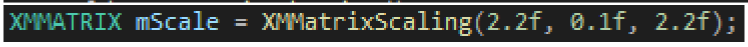
Solution:



```
ConstantBuffer cb;

XMMATRIX mScale = XMMatrixScaling(2.2f, 0.5f, 0.5f);
g_World = XMMatrixIdentity();
g_World *= mScale;

g_World *= XMMatrixRotationY(0.5f);
```



```
XMMATRIX mScale = XMMatrixScaling(0.5f, 2.2f, 0.5f);
```

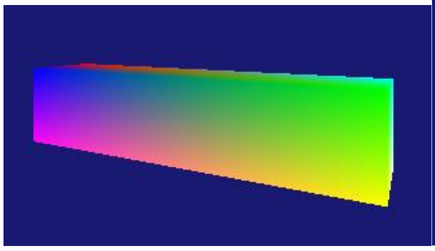


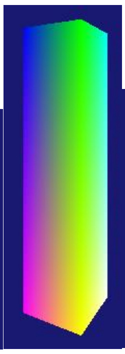
```
XMMATRIX mScale = XMMatrixScaling(2.2f, 0.1f, 2.2f);
```

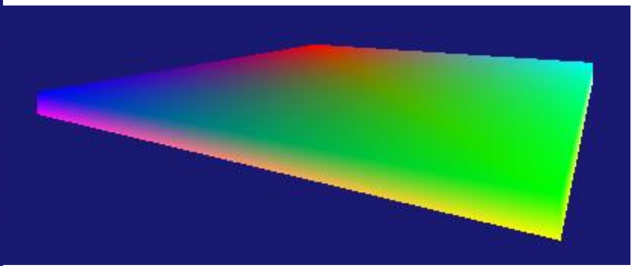
Test data:

n/a

Sample output:







Reflection:

These were really easy to do, the only challenge was getting the proportions right to make shapes like the examples

Metadata:

Scaling

Further information:

n/a

Q2. Exercise 2

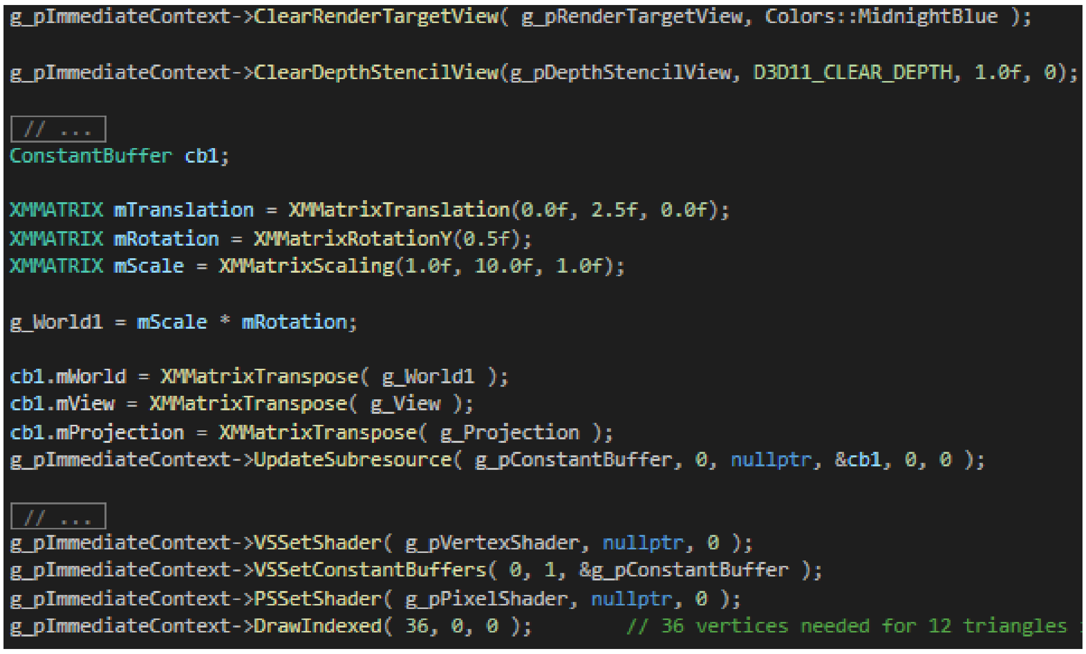
Question:

Perform scaling, translation, and rotation transformation to achieve the following effects:

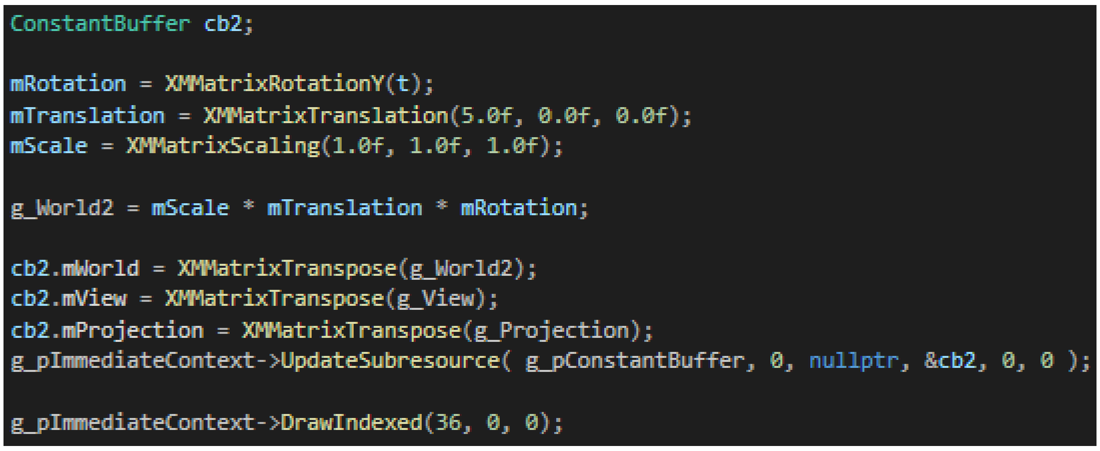
1. A cube rotates by a vertical rotation axis
2. Two cubes rotate by two different rotational axes with different rotational speeds respectively

Solution:

1.

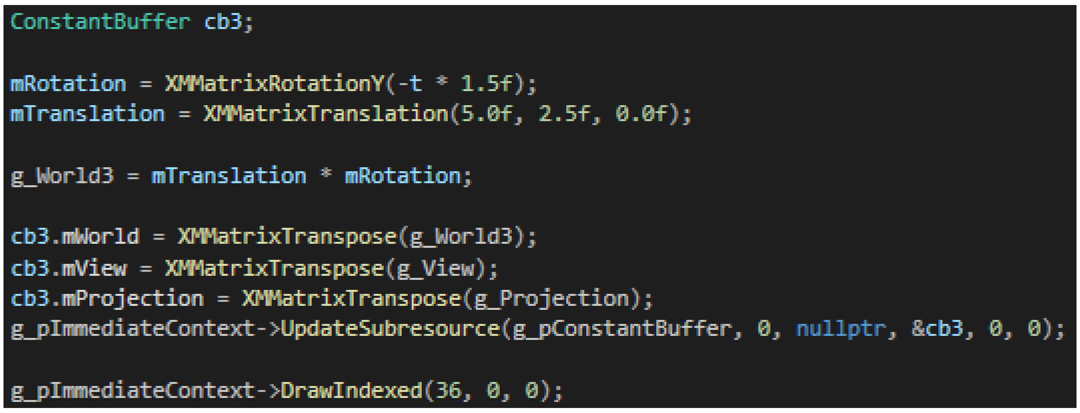


```
g_pImmediateContext->ClearRenderTargetView( g_pRenderTargetView, Colors::MidnightBlue );

g_pImmediateContext->ClearDepthStencilView(g_pDepthStencilView, D3D11_CLEAR_DEPTH, 1.0f, 0);

// ...
ConstantBuffer cb1;

XMMATRIX mTranslation = XMMatrixTranslation(0.0f, 2.5f, 0.0f);
XMMATRIX mRotation = XMMatrixRotationY(0.5f);
XMMATRIX mScale = XMMatrixScaling(1.0f, 10.0f, 1.0f);

g_World1 = mScale * mRotation;

cb1.mWorld = XMMatrixTranspose( g_World1 );
cb1.mView = XMMatrixTranspose( g_View );
cb1.mProjection = XMMatrixTranspose( g_Projection );
g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb1, 0, 0 );

// ...
g_pImmediateContext->VSSetShader( g_pVertexShader, nullptr, 0 );
g_pImmediateContext->VSSetConstantBuffers( 0, 1, &g_pConstantBuffer );
g_pImmediateContext->PSSetShader( g_pPixelShader, nullptr, 0 );
g_pImmediateContext->DrawIndexed( 36, 0, 0 );        // 36 vertices needed for 12 triangles
```



```
ConstantBuffer cb2;

mRotation = XMMatrixRotationY(t);
mTranslation = XMMatrixTranslation(5.0f, 0.0f, 0.0f);
mScale = XMMatrixScaling(1.0f, 1.0f, 1.0f);

g_World2 = mScale * mTranslation * mRotation;

cb2.mWorld = XMMatrixTranspose(g_World2);
cb2.mView = XMMatrixTranspose(g_View);
cb2.mProjection = XMMatrixTranspose(g_Projection);
g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb2, 0, 0 );

g_pImmediateContext->DrawIndexed(36, 0, 0);
```

2.



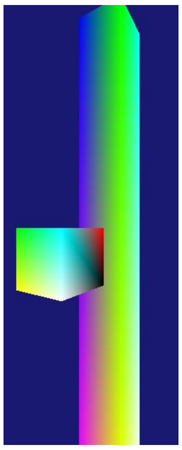
```
ConstantBuffer cb3;

mRotation = XMMatrixRotationY(-t * 1.5f);
mTranslation = XMMatrixTranslation(5.0f, 2.5f, 0.0f);

g_World3 = mTranslation * mRotation;

cb3.mWorld = XMMatrixTranspose(g_World3);
cb3.mView = XMMatrixTranspose(g_View);
cb3.mProjection = XMMatrixTranspose(g_Projection);
g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb3, 0, 0);

g_pImmediateContext->DrawIndexed(36, 0, 0);
```

Test data:

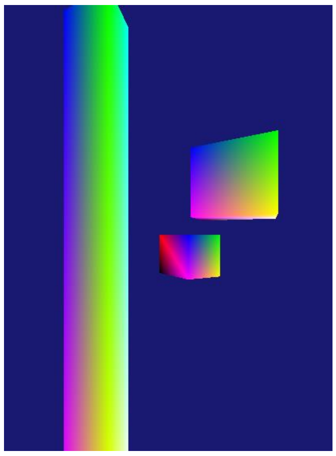
n/a

Sample output:

1.



2.



Reflection:

This was fairly simple to do once I realised that I needed to create a depth stencil to make the cubes go behind the pillar properly.

Metadata:

Cube around a pillar

Further information:

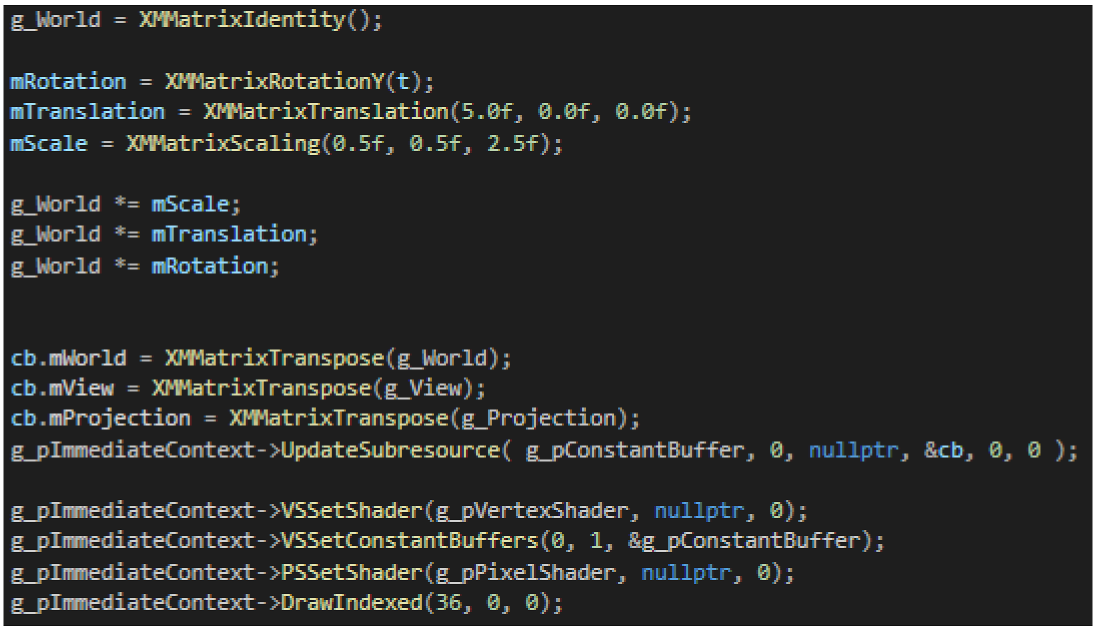
Transformation order is Scale the rotation then translation. However, different orders can be used for different effects

Q3. Exercise 3

Question:

scale the small rotating cube into a long-thin stick and rote the stick by a rotation axis such that the stick is always tangent to the rotation path.

Solution:

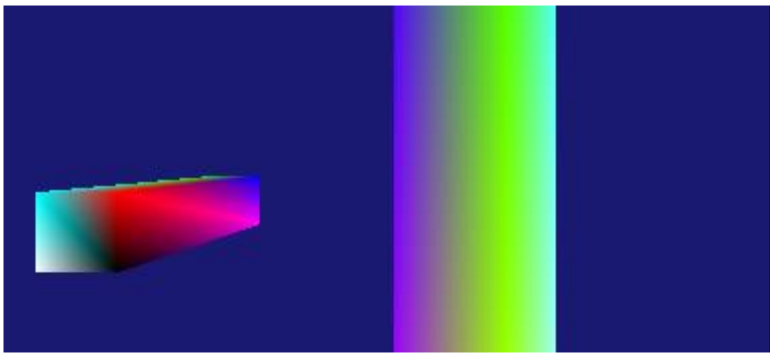


```
g_World = XMMatrixIdentity();

mRotation = XMMatrixRotationY(t);
mTranslation = XMMatrixTranslation(5.0f, 0.0f, 0.0f);
mScale = XMMatrixScaling(0.5f, 0.5f, 2.5f);

g_World *= mScale;
g_World *= mTranslation;
g_World *= mRotation;


cb.mWorld = XMMatrixTranspose(g_World);
cb.mView = XMMatrixTranspose(g_View);
cb.mProjection = XMMatrixTranspose(g_Projection);
g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb, 0, 0 );

g_pImmediateContext->VSSetShader(g_pVertexShader, nullptr, 0);
g_pImmediateContext->VSSetConstantBuffers(0, 1, &g_pConstantBuffer);
g_pImmediateContext->PSSetShader(g_pPixelShader, nullptr, 0);
g_pImmediateContext->DrawIndexed(36, 0, 0);
```

Test data:

n/a

Sample output:



Reflection:

This is the exact same as the last exercise except the scaling is different. Have I misunderstood what one of the exercises is asking for?

Metadata:

Tangent rotation

Further information:

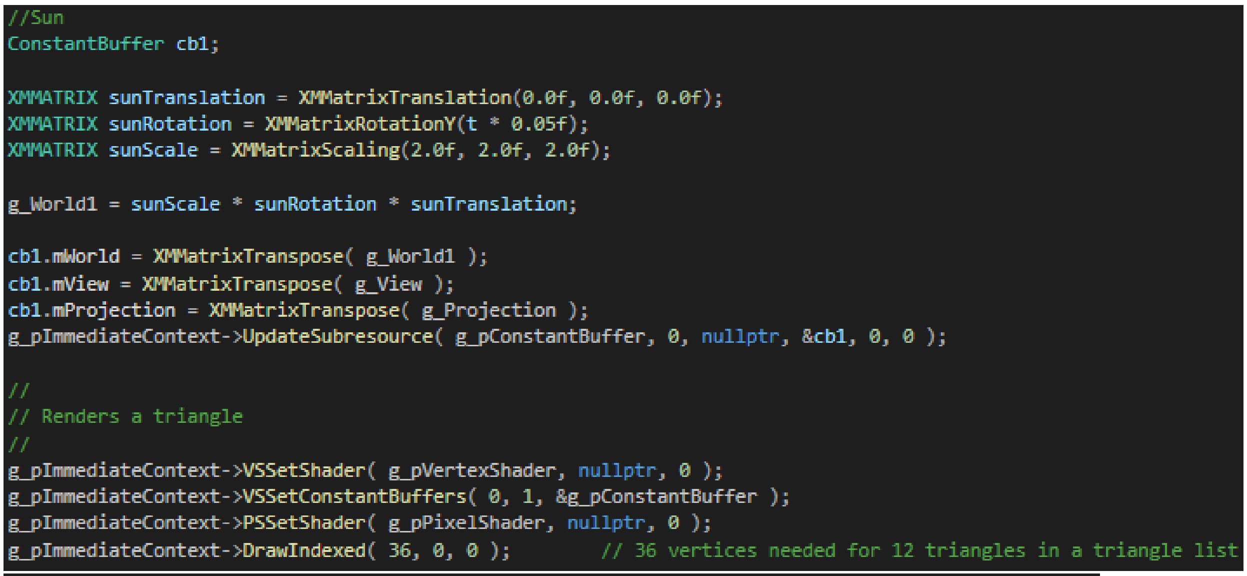
n/a

Q4. Exercise 4

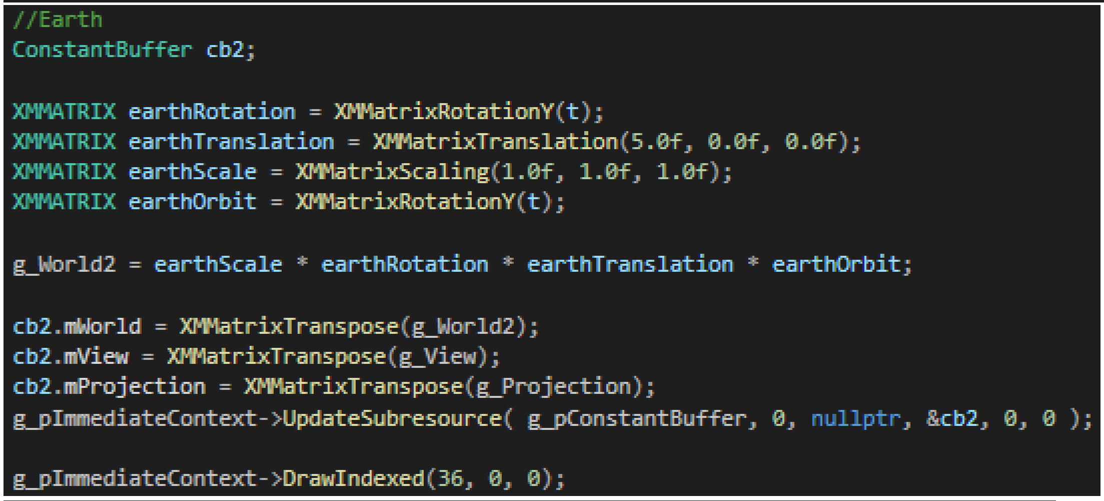
Question:

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

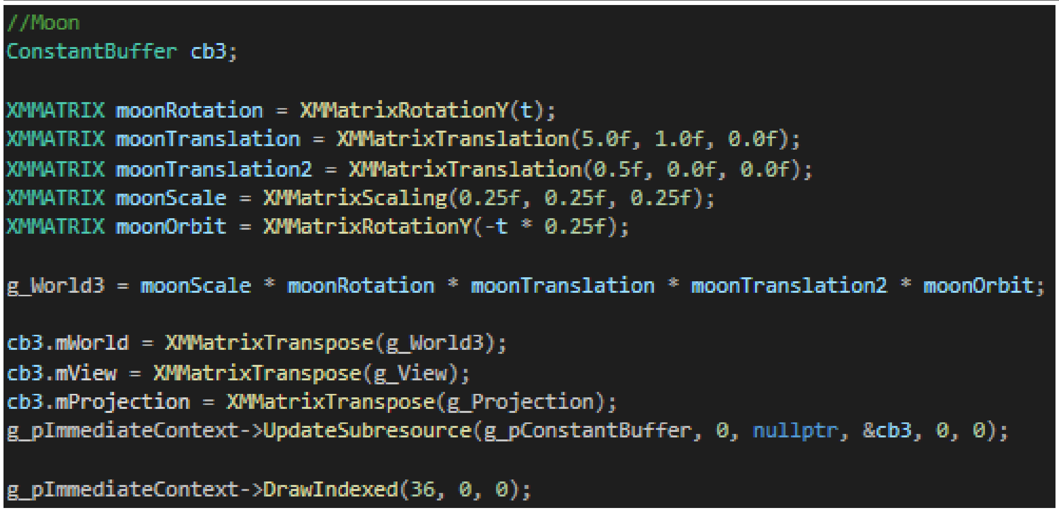
Solution:



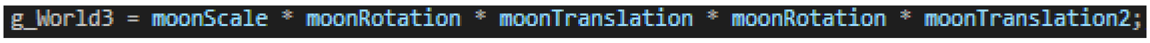
```
//Sun
ConstantBuffer cb1;

XMMATRIX sunTranslation = XMMatrixTranslation(0.0f, 0.0f, 0.0f);
XMMATRIX sunRotation = XMMatrixRotationY(t * 0.05f);
XMMATRIX sunScale = XMMatrixScaling(2.0f, 2.0f, 2.0f);

g_World1 = sunScale * sunRotation * sunTranslation;

cb1.mWorld = XMMatrixTranspose( g_World1 );
cb1.mView = XMMatrixTranspose( g_View );
cb1.mProjection = XMMatrixTranspose( g_Projection );
g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb1, 0, 0 );

//
// Renders a triangle
//
g_pImmediateContext->VSSetShader( g_pVertexShader, nullptr, 0 );
g_pImmediateContext->VSSetConstantBuffers( 0, 1, &g_pConstantBuffer );
g_pImmediateContext->PSSetShader( g_pPixelShader, nullptr, 0 );
g_pImmediateContext->DrawIndexed( 36, 0, 0 );        // 36 vertices needed for 12 triangles in a triangle list
```



```
//Earth
ConstantBuffer cb2;

XMMATRIX earthRotation = XMMatrixRotationY(t);
XMMATRIX earthTranslation = XMMatrixTranslation(5.0f, 0.0f, 0.0f);
XMMATRIX earthScale = XMMatrixScaling(1.0f, 1.0f, 1.0f);
XMMATRIX earthOrbit = XMMatrixRotationY(t);

g_World2 = earthScale * earthRotation * earthTranslation * earthOrbit;

cb2.mWorld = XMMatrixTranspose(g_World2);
cb2.mView = XMMatrixTranspose(g_View);
cb2.mProjection = XMMatrixTranspose(g_Projection);
g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb2, 0, 0 );

g_pImmediateContext->DrawIndexed(36, 0, 0);
```



```
//Moon
ConstantBuffer cb3;

XMMATRIX moonRotation = XMMatrixRotationY(t);
XMMATRIX moonTranslation = XMMatrixTranslation(5.0f, 1.0f, 0.0f);
XMMATRIX moonTranslation2 = XMMatrixTranslation(0.5f, 0.0f, 0.0f);
XMMATRIX moonScale = XMMatrixScaling(0.25f, 0.25f, 0.25f);
XMMATRIX moonOrbit = XMMatrixRotationY(-t * 0.25f);

g_World3 = moonScale * moonRotation * moonTranslation * moonTranslation2 * moonOrbit;

cb3.mWorld = XMMatrixTranspose(g_World3);
cb3.mView = XMMatrixTranspose(g_View);
cb3.mProjection = XMMatrixTranspose(g_Projection);
g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb3, 0, 0);

g_pImmediateContext->DrawIndexed(36, 0, 0);
```

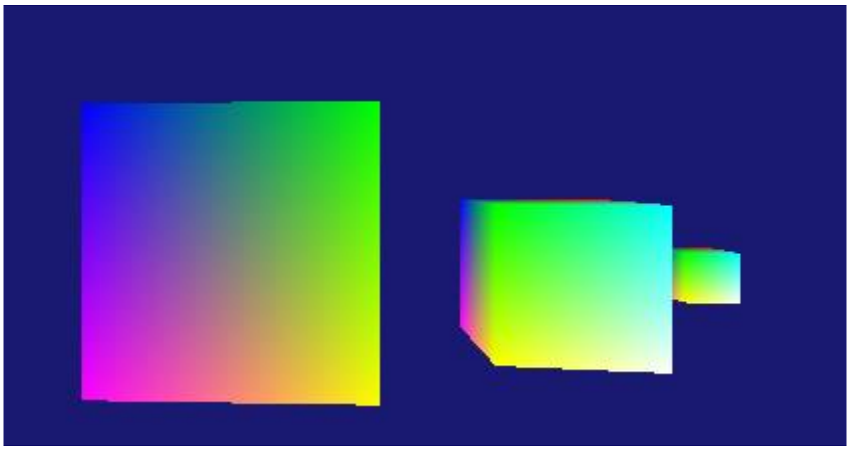


```
g_World3 = moonScale * moonRotation * moonTranslation * moonRotation * moonTranslation2;
```

Test data:

n/a

Sample output:



Reflection:

The sun and the earth were easy to do but the moon I can't quite figure out. I can get it to be the right size and have its own spin, but I can't get it to orbit the earth. I'm not sure how to make the origin point for the moons orbit be the centre of the earth cube.

My second attempt at getting the moon right looks like its work but it's hard to tell if it actually is.

Metadata:

Solar system

Further information:

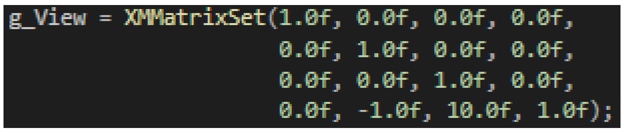
n/a

Q5. Exercise 5

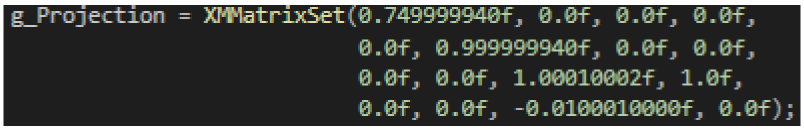
Question:

Manually set the view and g_Projection and observe if you get the same effect

Solution:



```
g_View = XMMatrixSet(1.0f, 0.0f, 0.0f, 0.0f,
                     0.0f, 1.0f, 0.0f, 0.0f,
                     0.0f, 0.0f, 1.0f, 0.0f,
                     0.0f, -1.0f, 10.0f, 1.0f);
```

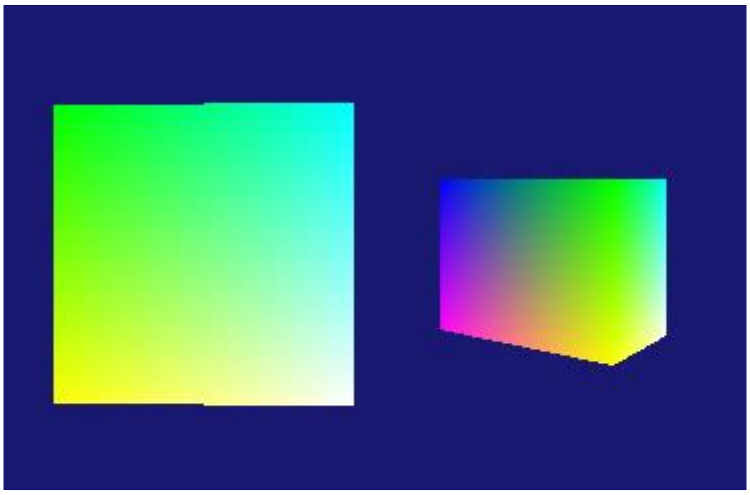


```
g_Projection = XMMatrixSet(0.749999940f, 0.0f, 0.0f, 0.0f,
                           0.0f, 0.999999940f, 0.0f, 0.0f,
                           0.0f, 0.0f, 1.00010002f, 1.0f,
                           0.0f, 0.0f, -0.0100010000f, 0.0f);
```

Test data:

n/a

Sample output:



Reflection:

I seem to get same effect but this could be because I copied the matrices that get produce by the XMMatrixLookAtLH() and XMMatrixPerspectiveFovLH() and hardcoded them.

Metadata:

Hardcoded matrices

Further information:

n/a

Week 4 – Lab 4

Date: 18/10/2022

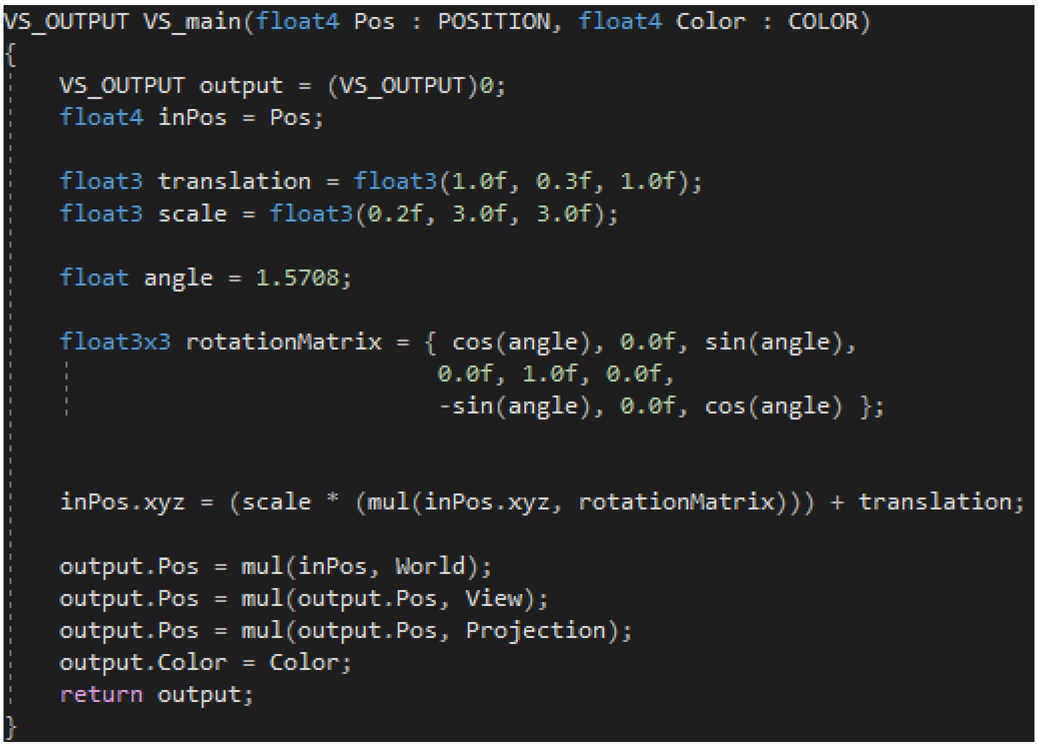
Q1. Create your own vertex shaders

Question:

1. Create a new vertex shader in "Tutorial04.fx" this vertex shader should translate by (1.0f, 0.3f, 1.0f) , scale by (0.2f, 3.0f, 3.0f) and rotate by the y-axis.
2. Create a new HLSL file and copy your new vertex shader into it then proceed to set it properties correctly.

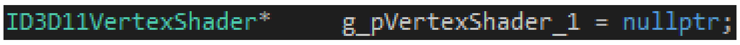
Solution:

1.

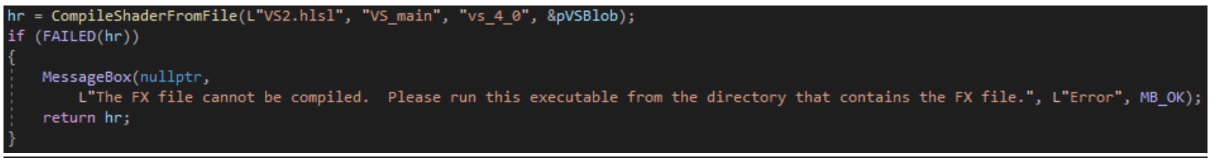


```
VS_OUTPUT VS_main(float4 Pos : POSITION, float4 Color : COLOR)
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    float4 inPos = Pos;

    float3 translation = float3(1.0f, 0.3f, 1.0f);
    float3 scale = float3(0.2f, 3.0f, 3.0f);

    float angle = 1.5708;

    float3x3 rotationMatrix = { cos(angle), 0.0f, sin(angle),
                                0.0f, 1.0f, 0.0f,
                                -sin(angle), 0.0f, cos(angle) };


    inPos.xyz = (scale * (mul(inPos.xyz, rotationMatrix))) + translation;

    output.Pos = mul(inPos, World);
    output.Pos = mul(output.Pos, View);
    output.Pos = mul(output.Pos, Projection);
    output.Color = Color;
    return output;
}
```

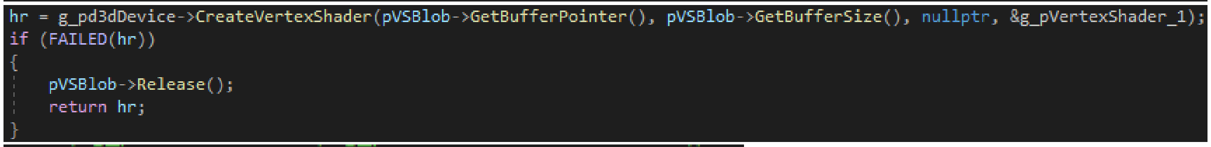
2.



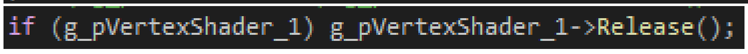
```
ID3D11VertexShader*      g_pVertexShader_1 = nullptr;
```



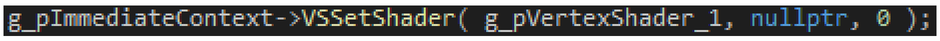
```
hr = CompileShaderFromFile(L"VS2.hlsl", "VS_main", "vs_4_0", &pVSBlob);
if (FAILED(hr))
{
    MessageBox(nullptr,
        L"The FX file cannot be compiled.  Please run this executable from the directory that contains the FX file.", L"Error", MB_OK);
    return hr;
}
```



```
hr = g_pd3dDevice->CreateVertexShader(pVSBlob->GetBufferPointer(), pVSBlob->GetBufferSize(), nullptr, &g_pVertexShader_1);
if (FAILED(hr))
{
    pVSBlob->Release();
    return hr;
}
```



```
if (g_pVertexShader_1) g_pVertexShader_1->Release();
```



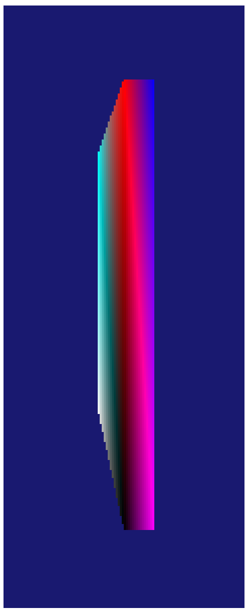
```
g_pImmediateContext->VSSetShader( g_pVertexShader_1, nullptr, 0 );
```

Test data:

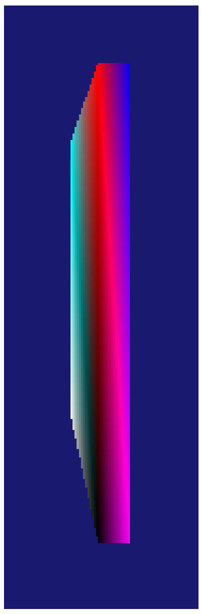
n/a

Sample output:

1.



2.



Reflection:

This took a while to do, I don't really understand HLSL very well so I need to practice more. Switching from an .fx file to a .hlsl file didn't seem to be much different.

Metadata:

HLSL

Further information:

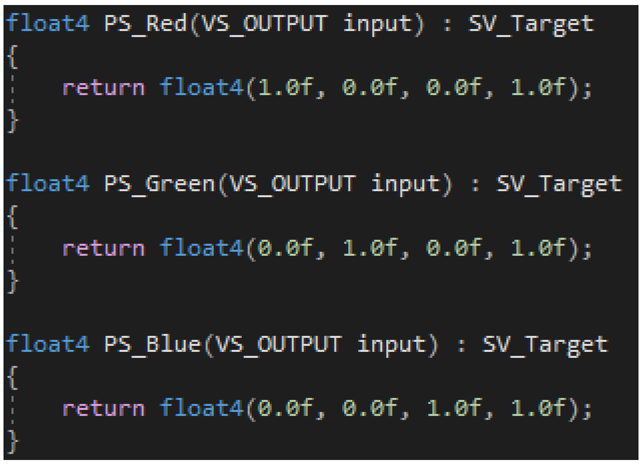
n/a

Q2. Create your own pixel shaders

Question:

Create pixel shaders to make 3 different coloured cubes

Solution:

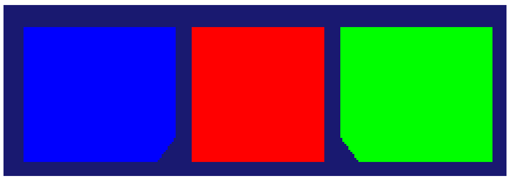


```
float4 PS_Red(VS_OUTPUT input) : SV_Target
{
    return float4(1.0f, 0.0f, 0.0f, 1.0f);
}

float4 PS_Green(VS_OUTPUT input) : SV_Target
{
    return float4(0.0f, 1.0f, 0.0f, 1.0f);
}

float4 PS_Blue(VS_OUTPUT input) : SV_Target
{
    return float4(0.0f, 0.0f, 1.0f, 1.0f);
}
```

Test data:

n/a

Sample output:



Reflection:

This was pretty simple to do. There is a lot of repetitive code to compile and create each pixel shader though, I'm sure that could improve on that by making a vertex shader and pixel shader class. I'm not sure why a chunk of the blue and green cube is missing, if I set the translations so each cube is right next to each other then it doesn't happen

Metadata:

Pixel shaders

Further information:

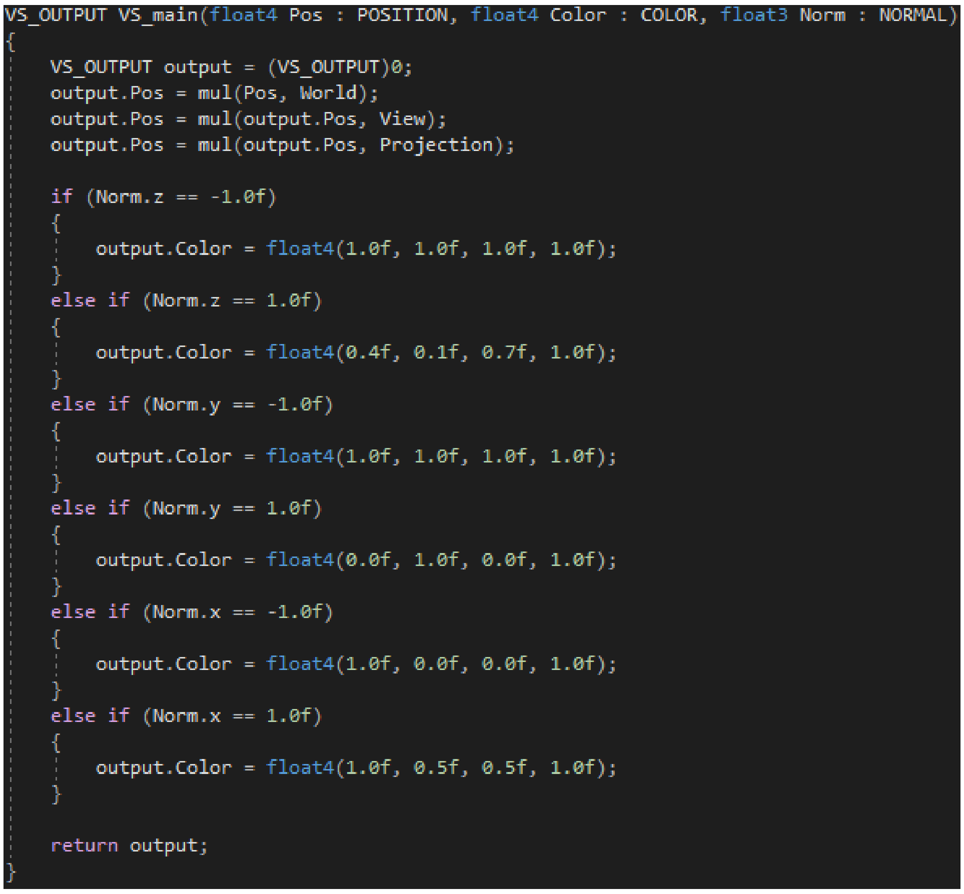
n/a

Q3. Cornell box in vertex shaders

Question:

Create Cornell box by using THREE different vertex shaders

Solution:

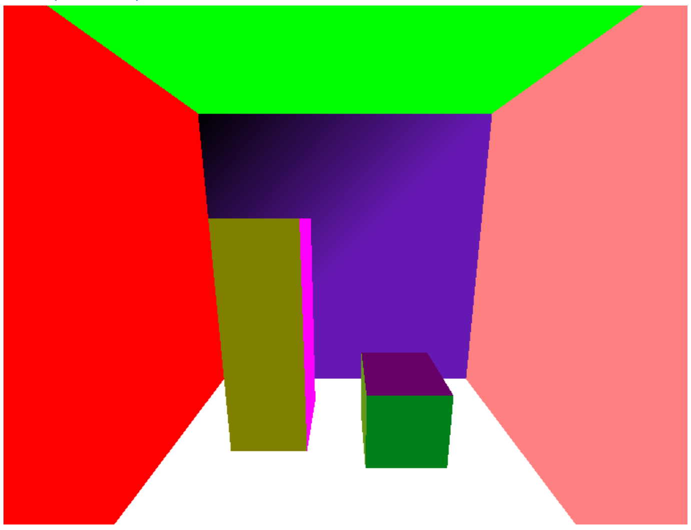


```
VS_OUTPUT VS_main(float4 Pos : POSITION, float4 Color : COLOR, float3 Norm : NORMAL)
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    output.Pos = mul(Pos, World);
    output.Pos = mul(output.Pos, View);
    output.Pos = mul(output.Pos, Projection);

    if (Norm.z == -1.0f)
    {
        output.Color = float4(1.0f, 1.0f, 1.0f, 1.0f);
    }
    else if (Norm.z == 1.0f)
    {
        output.Color = float4(0.4f, 0.1f, 0.7f, 1.0f);
    }
    else if (Norm.y == -1.0f)
    {
        output.Color = float4(1.0f, 1.0f, 1.0f, 1.0f);
    }
    else if (Norm.y == 1.0f)
    {
        output.Color = float4(0.0f, 1.0f, 0.0f, 1.0f);
    }
    else if (Norm.x == -1.0f)
    {
        output.Color = float4(1.0f, 0.0f, 0.0f, 1.0f);
    }
    else if (Norm.x == 1.0f)
    {
        output.Color = float4(1.0f, 0.5f, 0.5f, 1.0f);
    }

    return output;
}
```

Test data:

n/a

Sample output:



Reflection:

It doesn't look exactly like the example we were given but its close enough I think. For some reason the vertex shader won't colour the last vertex and I'm not sure why.

Metadata:

Cornell

Further information:

n/a

Q4. Define Model-View-Projection in vertex shader (optional)

Question:

Solution:

Test data:

Sample output:

Reflection:

As this is optional I will skip it for now as I am currently behind in the labs

Metadata:

Further information:

Q5. Vertex shader point cloud (Optional)

Question:

Solution:

Test data:

Sample output:

Reflection:

As this is optional I will skip it for now as I am currently behind in the labs

Metadata:

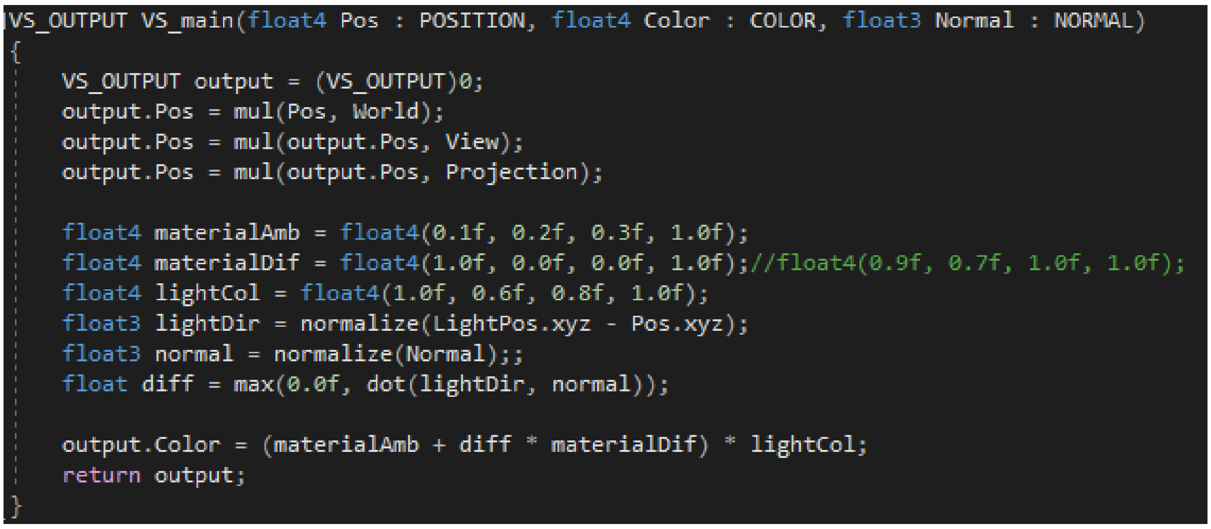
Further information:

Q6. Per-vertex diffuse lighting

Question:

Create diffuse lighting in your vertex shader. Then modify the material reflection coefficient so that only red light is reflected

Solution:

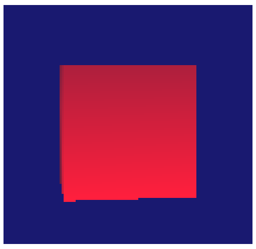


```
VS_OUTPUT VS_main(float4 Pos : POSITION, float4 Color : COLOR, float3 Normal : NORMAL)
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    output.Pos = mul(Pos, World);
    output.Pos = mul(output.Pos, View);
    output.Pos = mul(output.Pos, Projection);

    float4 materialAmb = float4(0.1f, 0.2f, 0.3f, 1.0f);
    float4 materialDif = float4(1.0f, 0.0f, 0.0f, 1.0f);//float4(0.9f, 0.7f, 1.0f, 1.0f);
    float4 lightCol = float4(1.0f, 0.6f, 0.8f, 1.0f);
    float3 lightDir = normalize(LightPos.xyz - Pos.xyz);
    float3 normal = normalize(Normal);;
    float diff = max(0.0f, dot(lightDir, normal));

    output.Color = (materialAmb + diff * materialDif) * lightCol;
    return output;
}
```

Test data:

n/a

Sample output:



Reflection:

This was pretty straight-forward to do there isn't much to comment on. There was quite a lot of set up work to get this to work and I'm not sure if I really understand the depth-stencil texture and depth-stencil view.

Metadata:

Light

Further information:

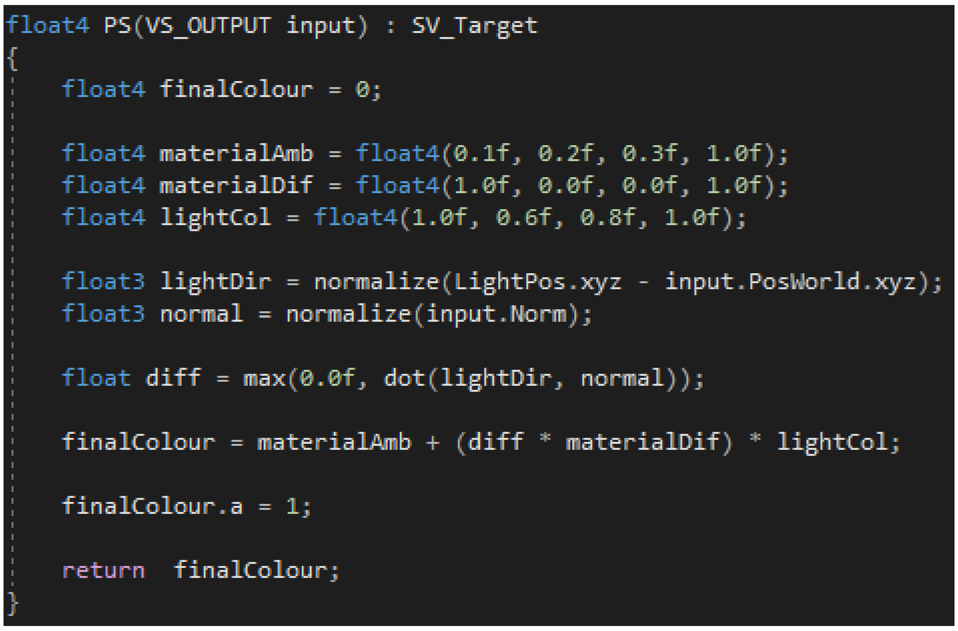
n/a

Q7. Per-pixel diffuse lighting

Question:

Create diffuse lighting in the pixel shader

Solution:

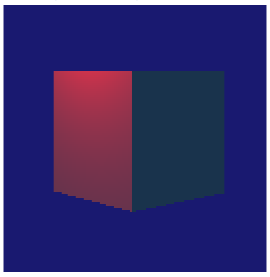


```
float4 PS(VS_OUTPUT input) : SV_Target
{
    float4 finalColour = 0;

    float4 materialAmb = float4(0.1f, 0.2f, 0.3f, 1.0f);
    float4 materialDif = float4(1.0f, 0.0f, 0.0f, 1.0f);
    float4 lightCol = float4(1.0f, 0.6f, 0.8f, 1.0f);

    float3 lightDir = normalize(LightPos.xyz - input.PosWorld.xyz);
    float3 normal = normalize(input.Norm);

    float diff = max(0.0f, dot(lightDir, normal));

    finalColour = materialAmb + (diff * materialDif) * lightCol;

    finalColour.a = 1;

    return  finalColour;
}
```

Test data:

n/a

Sample output:



Reflection:

Finally got this done it turns out I wasn’t passing the light position to the pixel shader. Lighting and Shader programming is still really confusing to me though

Metadata:

Pixel lighting

Further information:

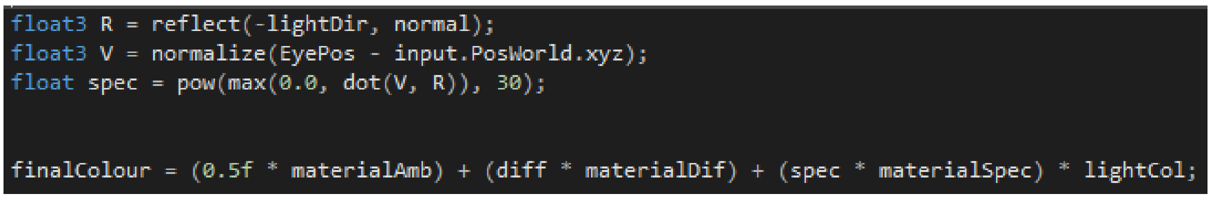
n/a

Q8. Per-pixel specular lighting

Question:

Create specular lighting in the pixel shader

Solution:

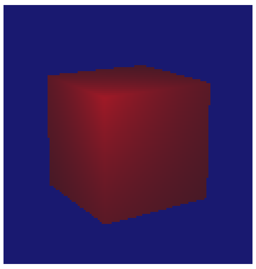


```
float3 R = reflect(-lightDir, normal);
float3 V = normalize(EyePos - input.PosWorld.xyz);
float spec = pow(max(0.0, dot(V, R)), 30);


finalColour = (0.5f * materialAmb) + (diff * materialDif) + (spec * materialSpec) * lightCol;
```

Test data:

n/a

Sample output:



Reflection:

This was fairly easy to do now that I have my pixel shader lighting working. I need to remember that with specular light that on this line: `float spec = pow(max(0.0, dot(V, R)), 30);` the 30 represents shininess of the surface 1 being most shiny 200 being the least

Metadata:

Specular Pixel

Further information:

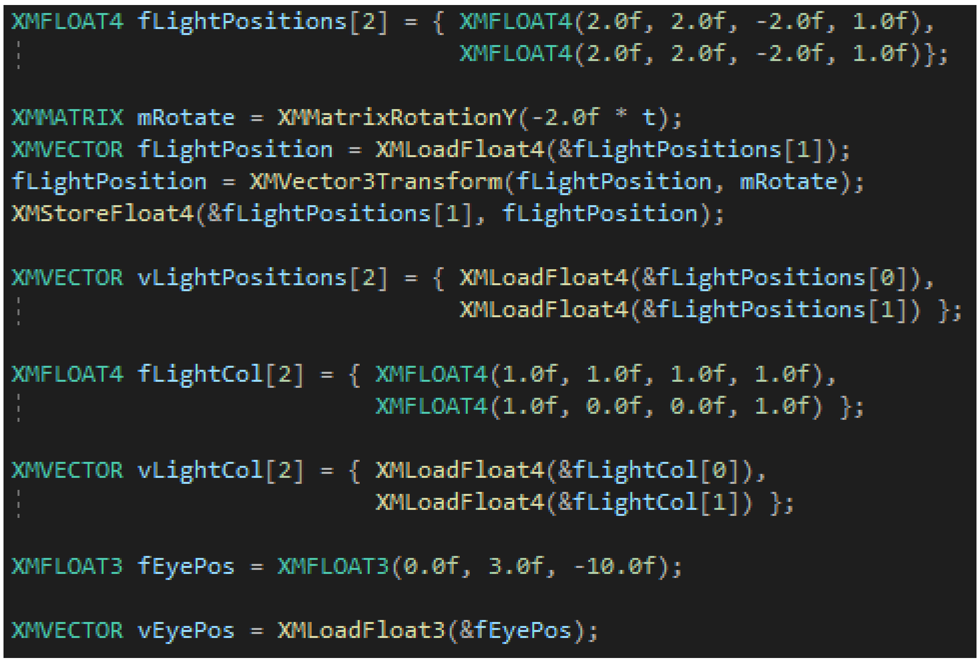
n/a

Q9. Multiple materials and light sources

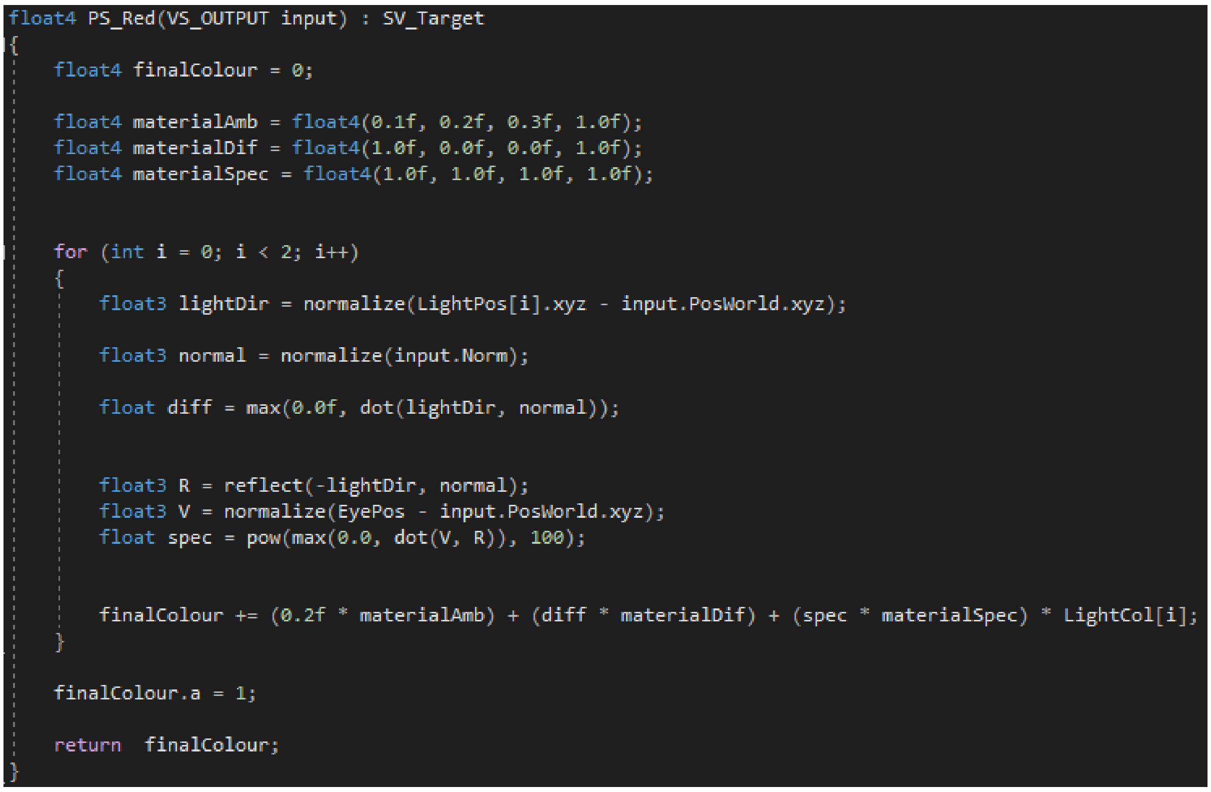
Question:

Draw three cube objects with three different surface material properties. Illuminate these cubes with two light sources: one is a static white light; the other is a red light rotating dynamically by the y-axis.

Solution:



```
XMFLOAT4 fLightPositions[2] = { XMFLOAT4(2.0f, 2.0f, -2.0f, 1.0f),
                                XMFLOAT4(2.0f, 2.0f, -2.0f, 1.0f)};

XMMATRIX mRotate = XMMatrixRotationY(-2.0f * t);
XMVECTOR fLightPosition = XMLoadFloat4(&fLightPositions[1]);
fLightPosition = XMVector3Transform(fLightPosition, mRotate);
XMStoreFloat4(&fLightPositions[1], fLightPosition);

XMVECTOR vLightPositions[2] = { XMLoadFloat4(&fLightPositions[0]),
                                XMLoadFloat4(&fLightPositions[1]) };

XMFLOAT4 fLightCol[2] = { XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f),
                          XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) };

XMVECTOR vLightCol[2] = { XMLoadFloat4(&fLightCol[0]),
                          XMLoadFloat4(&fLightCol[1]) };

XMFLOAT3 fEyePos = XMFLOAT3(0.0f, 3.0f, -10.0f);

XMVECTOR vEyePos = XMLoadFloat3(&fEyePos);
```

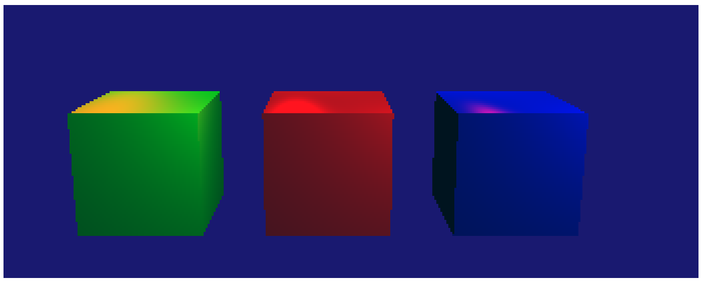


```
float4 PS_Red(VS_OUTPUT input) : SV_Target
{
    float4 finalColour = 0;

    float4 materialAmb = float4(0.1f, 0.2f, 0.3f, 1.0f);
    float4 materialDif = float4(1.0f, 0.0f, 0.0f, 1.0f);
    float4 materialSpec = float4(1.0f, 1.0f, 1.0f, 1.0f);


    for (int i = 0; i < 2; i++)
    {
        float3 lightDir = normalize(LightPos[i].xyz - input.PosWorld.xyz);

        float3 normal = normalize(input.Norm);

        float diff = max(0.0f, dot(lightDir, normal));


        float3 R = reflect(-lightDir, normal);
        float3 V = normalize(EyePos - input.PosWorld.xyz);
        float spec = pow(max(0.0, dot(V, R)), 100);


        finalColour += (0.2f * materialAmb) + (diff * materialDif) + (spec * materialSpec) * LightCol[i];
    }

    finalColour.a = 1;

    return  finalColour;
}
```

Test data:

n/a

Sample output:



Reflection:

I'm not completely confident that I have done this correctly. Based on what I know and by looking at tutorial06 I think I have successfully done it.

Metadata:

n/a

Further information:

Can you confirm if I have implemented this correctly? and if I haven't can you point out where I went wrong please. For some reason I'm struggling to understand lighting fully.

Week 5 – Lab 5

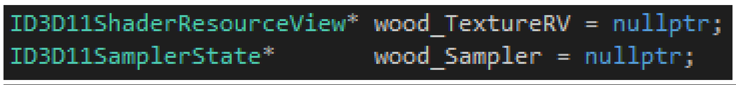
Date: 26/10/2022

Q1. A wooden box

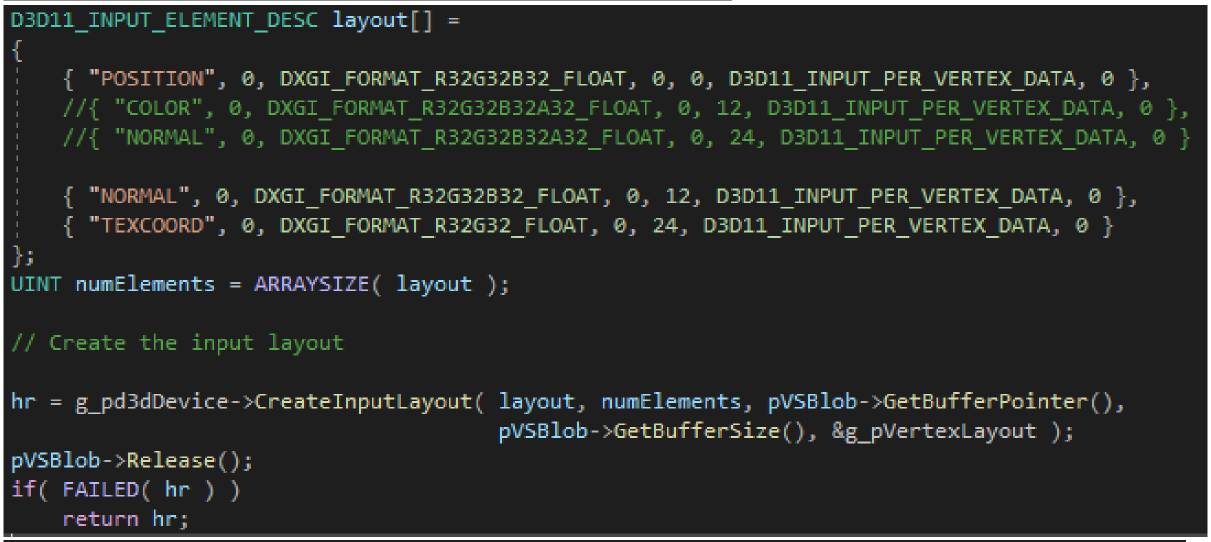
Question:

Give your cube a wood texture

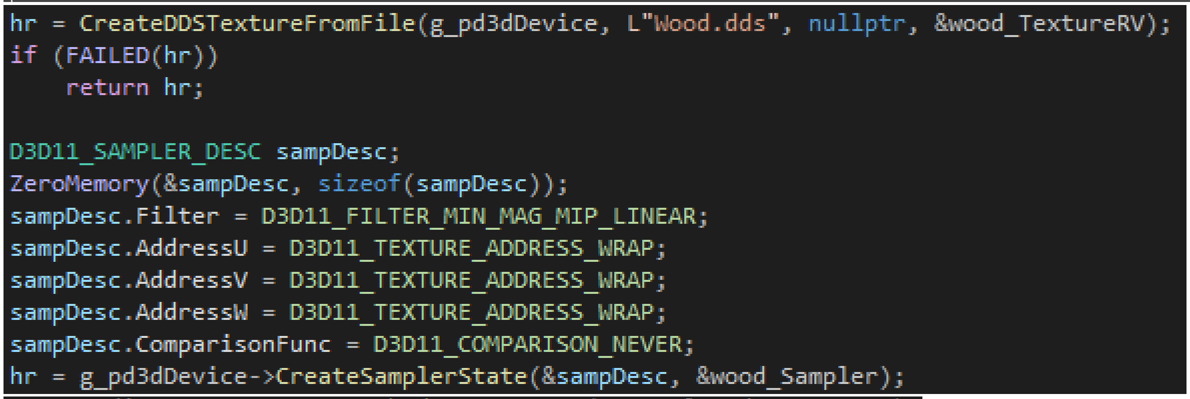
Solution:



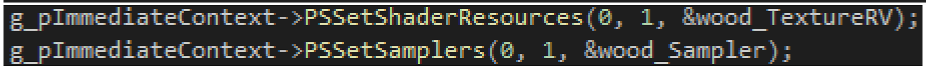
```
ID3D11ShaderResourceView* wood_TextureRV = nullptr;
ID3D11SamplerState*       wood_Sampler = nullptr;
```



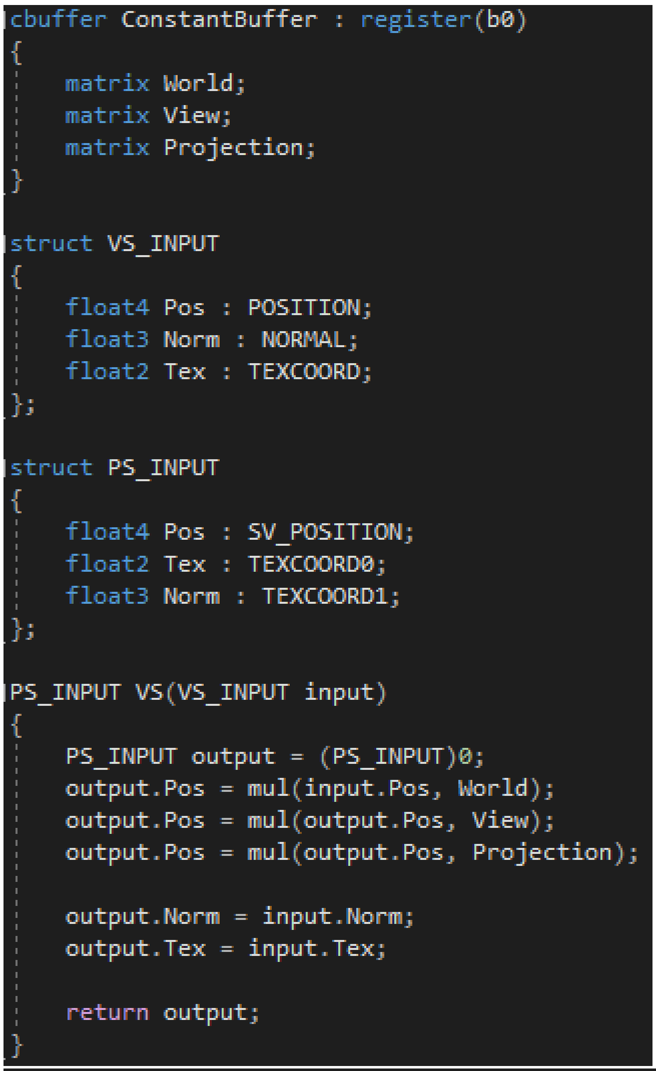
```
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    //{ "COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    //{ "NORMAL", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 24, D3D11_INPUT_PER_VERTEX_DATA, 0 }

    { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 24, D3D11_INPUT_PER_VERTEX_DATA, 0 }
};
UINT numElements = ARRAYSIZE( layout );

// Create the input layout

hr = g_pd3dDevice->CreateInputLayout( layout, numElements, pVSBlob->GetBufferPointer(),
                                      pVSBlob->GetBufferSize(), &g_pVertexLayout );
pVSBlob->Release();
if( FAILED( hr ) )
    return hr;
```



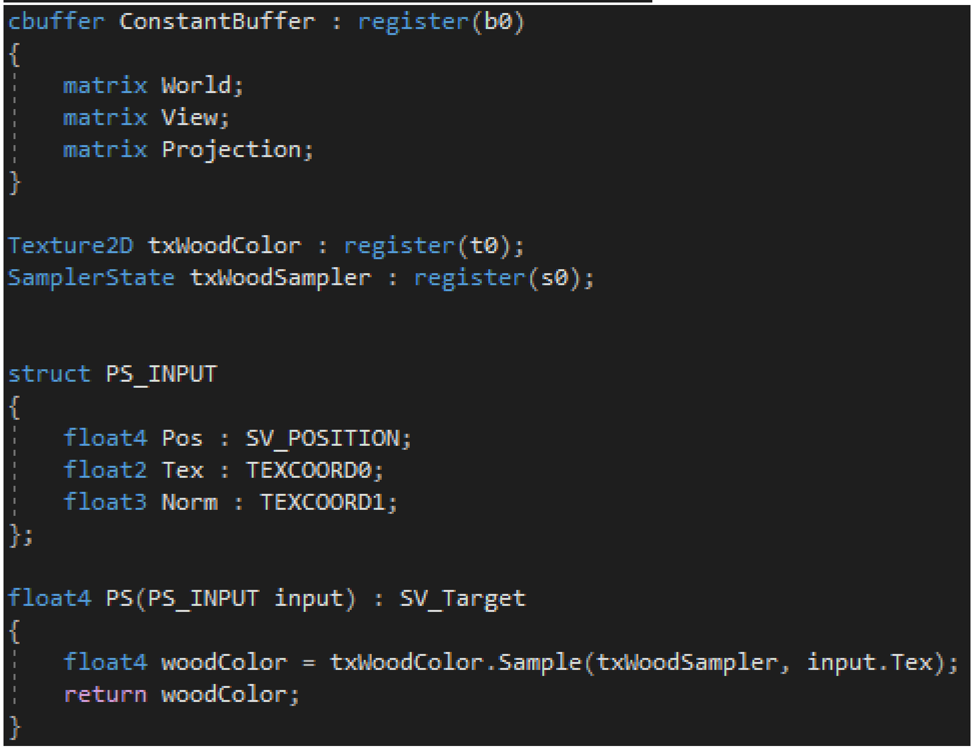
```
hr = CreateDDSTextureFromFile(g_pd3dDevice, L"Wood.dds", nullptr, &wood_TextureRV);
if (FAILED(hr))
    return hr;

D3D11_SAMPLER_DESC sampDesc;
ZeroMemory(&sampDesc, sizeof(sampDesc));
sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
sampDesc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;
hr = g_pd3dDevice->CreateSamplerState(&sampDesc, &wood_Sampler);
```



```
g_pImmediateContext->PSSetShaderResources(0, 1, &wood_TextureRV);
g_pImmediateContext->PSSetSamplers(0, 1, &wood_Sampler);
```



```
cbuffer ConstantBuffer : register(b0)
{
    matrix World;
    matrix View;
    matrix Projection;
}

struct VS_INPUT
{
    float4 Pos : POSITION;
    float3 Norm : NORMAL;
    float2 Tex : TEXCOORD;
};

struct PS_INPUT
{
    float4 Pos : SV_POSITION;
    float2 Tex : TEXCOORD0;
    float3 Norm : TEXCOORD1;
};

PS_INPUT VS(VS_INPUT input)
{
    PS_INPUT output = (PS_INPUT)0;
    output.Pos = mul(input.Pos, World);
    output.Pos = mul(output.Pos, View);
    output.Pos = mul(output.Pos, Projection);

    output.Norm = input.Norm;
    output.Tex = input.Tex;

    return output;
}
```

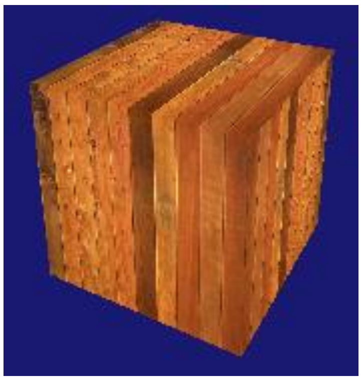


```
cbuffer ConstantBuffer : register(b0)
{
    matrix World;
    matrix View;
    matrix Projection;
}

Texture2D txWoodColor : register(t0);
SamplerState txWoodSampler : register(s0);


struct PS_INPUT
{
    float4 Pos : SV_POSITION;
    float2 Tex : TEXCOORD0;
    float3 Norm : TEXCOORD1;
};

float4 PS(PS_INPUT input) : SV_Target
{
    float4 woodColor = txWoodColor.Sample(txWoodSampler, input.Tex);
    return woodColor;
}
```

Test data:

n/a

Sample output:



Reflection:

This was pretty straight-forward the only issue I had was that the layout creation was failing but I eventually found out I had just misspelled one of the matching variables in the vertex shader.

Metadata:

Wooden cube

Further information:

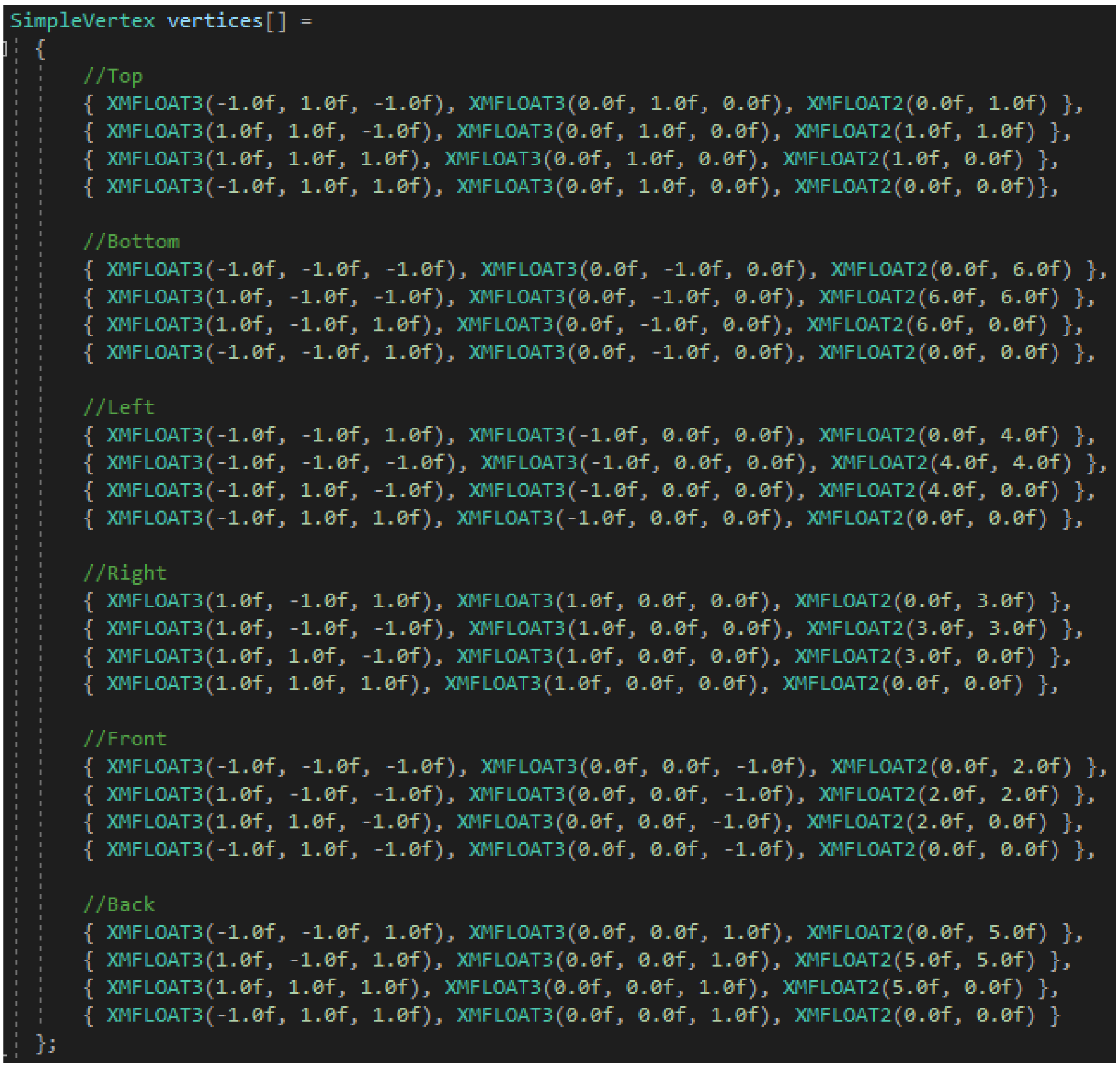
n/a

Q2. Texture wrapping mode

Question:

Create a texture-mapped cube using the coin texture

Solution:

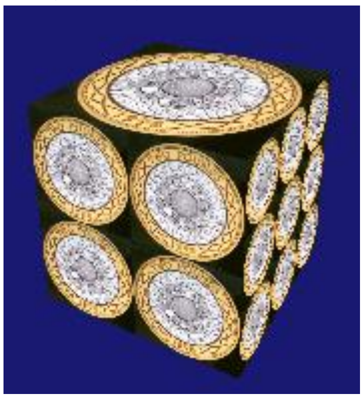


```
SimpleVertex vertices[] =
{
    //Top
    { XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT2(0.0f, 1.0f) },
    { XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT2(1.0f, 1.0f) },
    { XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f), XMFLOAT2(0.0f, 0.0f)},

    //Bottom
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT2(0.0f, 6.0f) },
    { XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT2(6.0f, 6.0f) },
    { XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT2(6.0f, 0.0f) },
    { XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f), XMFLOAT2(0.0f, 0.0f) },

    //Left
    { XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f), XMFLOAT2(0.0f, 4.0f) },
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f), XMFLOAT2(4.0f, 4.0f) },
    { XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f), XMFLOAT2(4.0f, 0.0f) },
    { XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f), XMFLOAT2(0.0f, 0.0f) },

    //Right
    { XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f), XMFLOAT2(0.0f, 3.0f) },
    { XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f), XMFLOAT2(3.0f, 3.0f) },
    { XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f), XMFLOAT2(3.0f, 0.0f) },
    { XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f), XMFLOAT2(0.0f, 0.0f) },

    //Front
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT2(0.0f, 2.0f) },
    { XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT2(2.0f, 2.0f) },
    { XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT2(2.0f, 0.0f) },
    { XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) },

    //Back
    { XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 5.0f) },
    { XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f), XMFLOAT2(5.0f, 5.0f) },
    { XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f), XMFLOAT2(5.0f, 0.0f) },
    { XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f) }
};
```

Test data:

n/a

Sample output:



Reflection:

This was simple but I don't really understand the other types of texture addresses or when I might want to use them

Metadata:

Texture addresses

Further information:

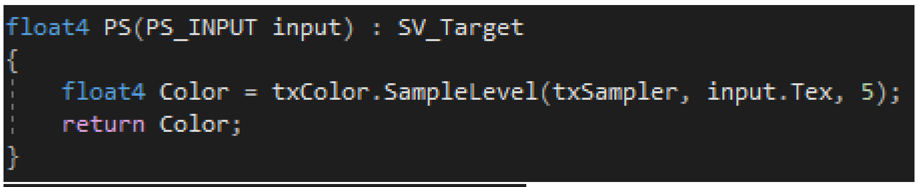
n/a

Q3. Mipmapping

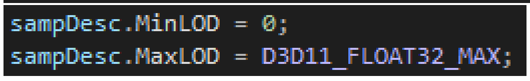
Question:

Create mipmaps of a loaded texture and use HLSL sampleLevel( ) to map the cube with different level of mipmaps

Solution:



```
float4 PS(PS_INPUT input) : SV_Target
{
    float4 Color = txColor.SampleLevel(txSampler, input.Tex, 5);
    return Color;
}
```

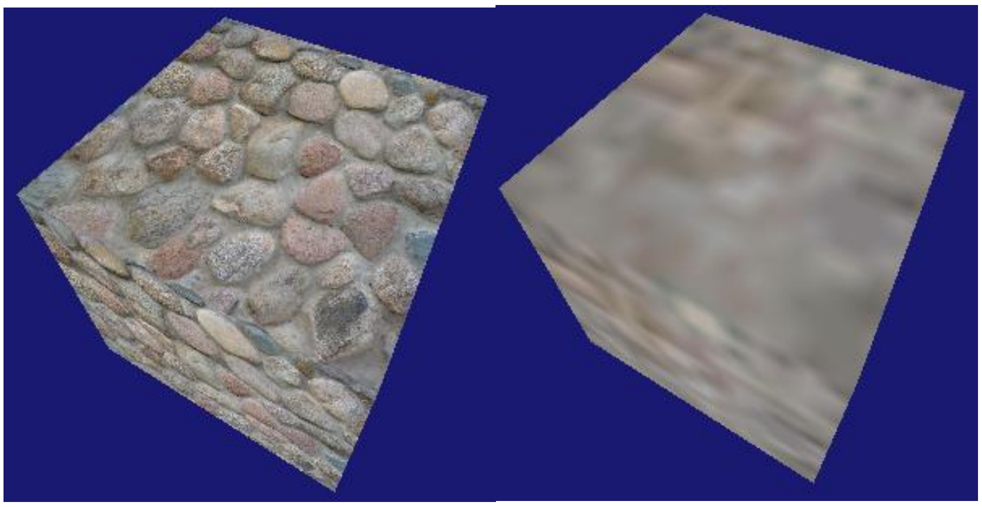


```
sampDesc.MinLOD = 0;
sampDesc.MaxLOD = D3D11_FLOAT32_MAX;
```

Test data:

n/a

Sample output:



Reflection:

This took way too long to do; I couldn't get it to work but then after trying it in tutorial 7 and comparing the code I realised I had not set somethings in the sampler description properly.

Metadata:

Mipmaps

Further information:

n/a

Q4. Texture filtering techniques

Question:

Scale the cube along the view direction to create a long rectangular object. Using different filtering techniques to deal with the minification and magnification issues on texture mapping and observe how visual quality of the rendered image is being changed.

Solution:

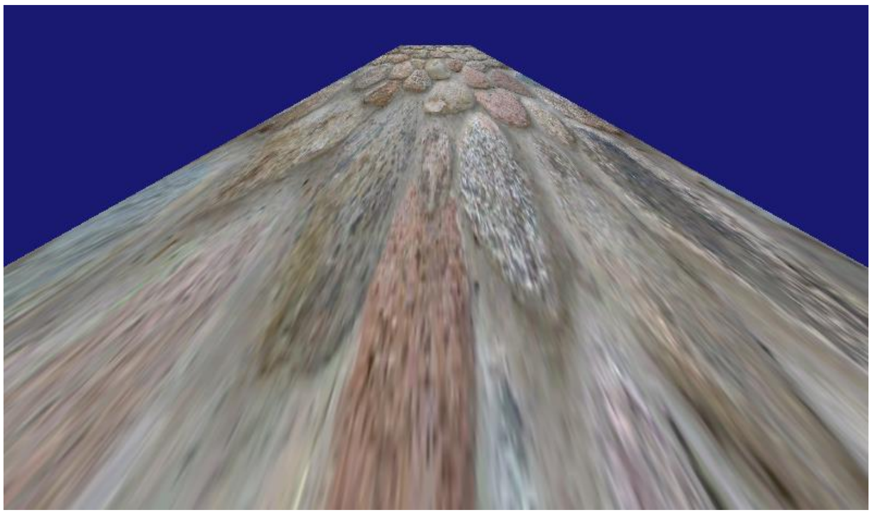
1. `sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;`
2. `sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_POINT;`
3. `sampDesc.Filter = D3D11_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR;`
4. `sampDesc.Filter = D3D11_FILTER_ANISOTROPIC;`

Test data:

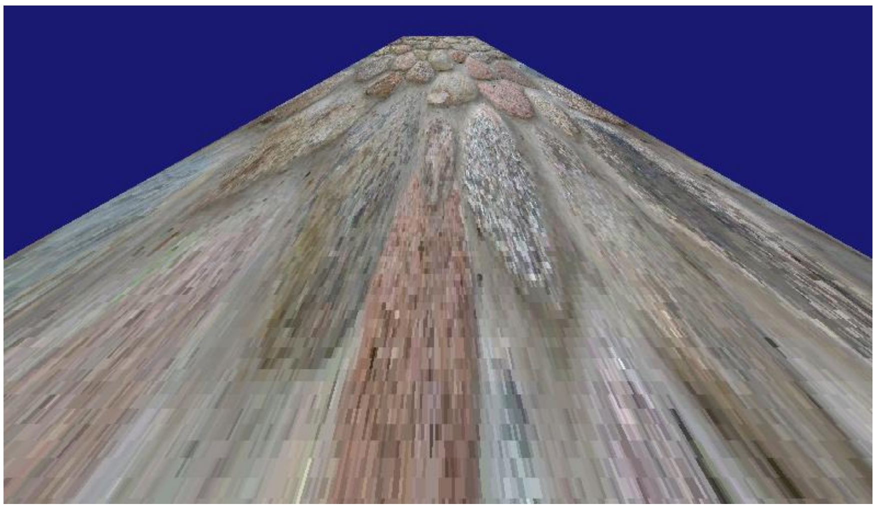
n/a

Sample output:

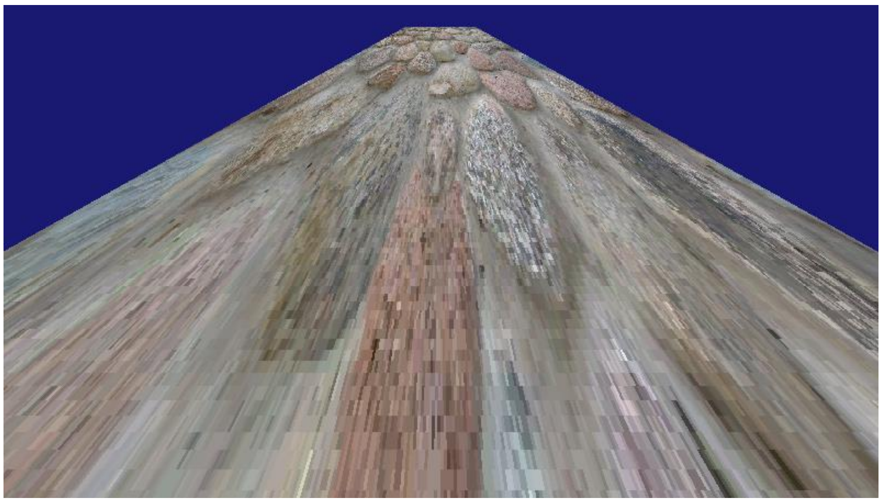
1.



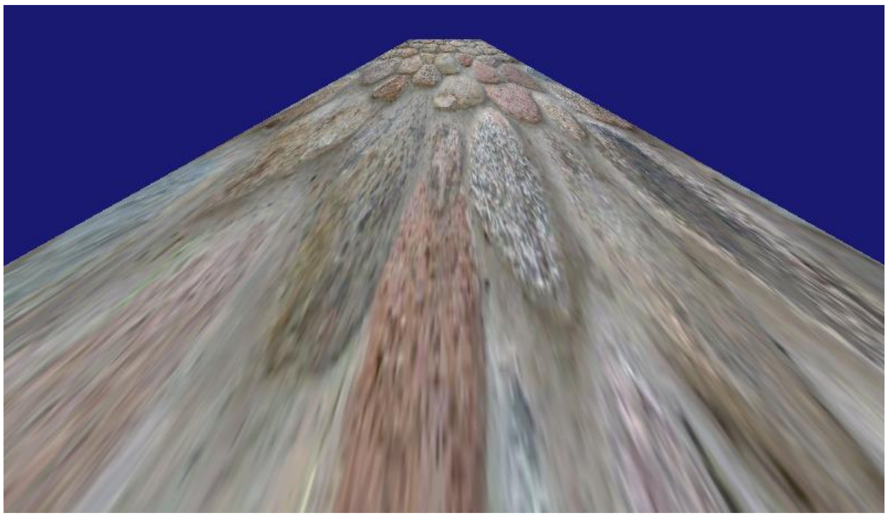
2.



3.



4.



Reflection:

2 and 3 seem to reduce the quality of the texture and make it much grainier. 1 and 4 seem to be pretty similar, 1 might be a little bit sharper. I'm not sure on the use cases for each filter I guess it just depends on the effect you are trying to achieve

Metadata:

Filters

Further information:

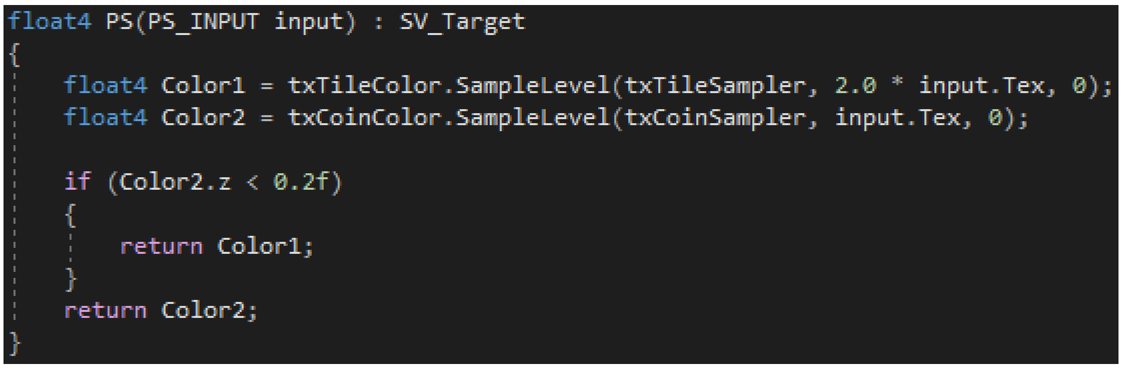
n/a

Q5. Multiple texturing

Question:

Create a cube that uses the tile with the coin texture layered on top.

Solution:

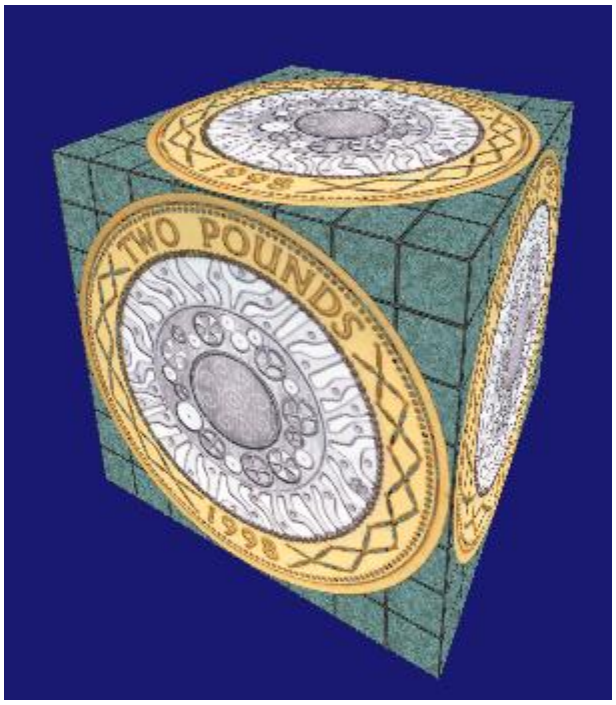


```
float4 PS(PS_INPUT input) : SV_Target
{
    float4 Color1 = txTileColor.SampleLevel(txTileSampler, 2.0 * input.Tex, 0);
    float4 Color2 = txCoinColor.SampleLevel(txCoinSampler, input.Tex, 0);

    if (Color2.z < 0.2f)
    {
        return Color1;
    }
    return Color2;
}
```

Test data:

n/a

Sample output:



Reflection:

This took a lot longer than it should of. I was trying to test the alpha of the texture by looking in the 4th float of a float4 but since the texture is 2-D the alpha is actually stored in the 3rd float of a float4. I think that's right. Please correct me if I am wrong because I'm not full sure how I managed to achieve the desired effect.

Metadata:

Multiple Textures

Further information:

n/a

C++ Programming & Design (700120) Labs

Week 1 – Lab A

Date: 27th Sep 2022

Q1. Hello World

Question:

Locate the Solution Explorer within Visual Studio and select the HelloWorld project. Right click on this project and select Build. This should compile and link the project. Now run the HelloWorld program.

Change between Debug and Release mode. Compile again and rerun the program.

Solution:

```cpp
#include <iostream>

int main(int, char**) {
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

Test data:

n/a

Sample output:

n/a

Reflection:

This is programming 101

Metadata:

Hello World

Further information:

What is purpose are the parameters in main?

Q2. Console Window

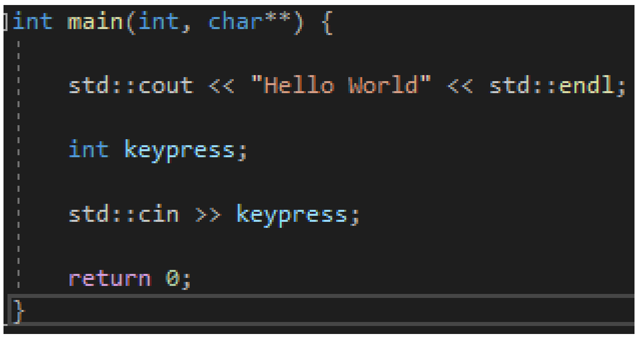
Question:

Delay the termination of the program by adding the following two lines to the end of your code:

int keypress;

std::cin >> keypress;

Solution:

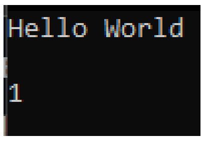


```
int main(int, char**) {

    std::cout << "Hello World" << std::endl;

    int keypress;

    std::cin >> keypress;

    return 0;
}
```

Test data:

n/a

Sample output:



```
Hello World

1
```

Reflection:

This just stops the program from terminating as soon as the program what finished running.

Metadata:

Delay termination

Further information:

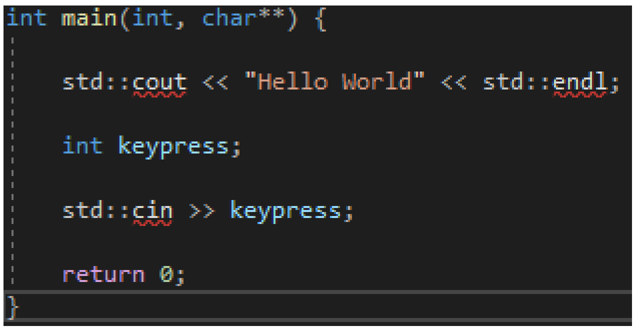
Is this something that’s only really done for educational purposes?

Q3. Includes

Question:

Remove “#include <iostream>” then compile, what is the effect>

Solution:



```
int main(int, char**) {

    std::cout << "Hello World" << std::endl;

    int keypress;

    std::cin >> keypress;

    return 0;
}
```

Test data:

n/a

Sample output:

n/a

Reflection:

cin and cout can’t be used because the program doesn’t have access to the iostream

Metadata:

includes

Further information:

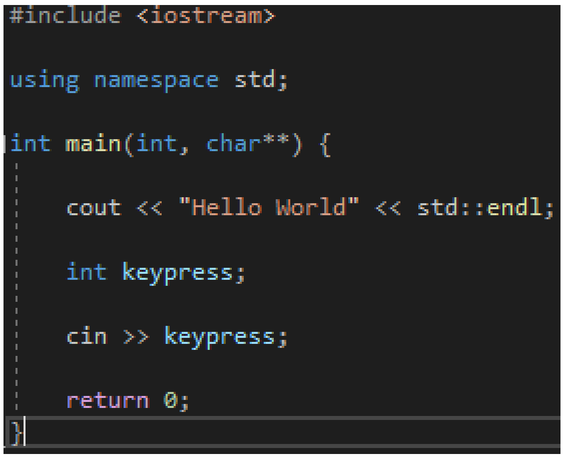
n/a

Q4. Namespace

Question:

Add “using namespace std;” and remove all “std::”

Solution:



```
#include <iostream>

using namespace std;

int main(int, char**) {

    cout << "Hello World" << std::endl;

    int keypress;

    cin >> keypress;

    return 0;
}
```

Test data:

n/a

Sample output:

n/a

Reflection:

I thought that visual studio would make you get rid of the “std::” to compile if you as using the std namespace but apparently not

Metadata:

Namespace std

Further information:

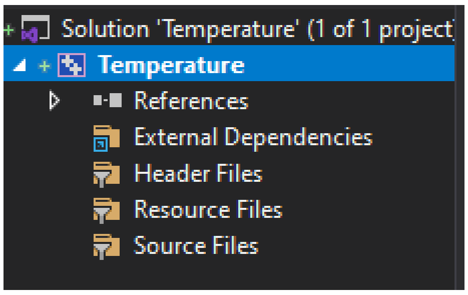
n/a

Q5. Create a new project

Question:

Create a new Visual C++ Empty project called “Temperature”

Solution:



Test data:

n/a

Sample output:

n/a

Reflection:

Very straight-forward, nothing to comment on

Metadata:

New project

Further information:

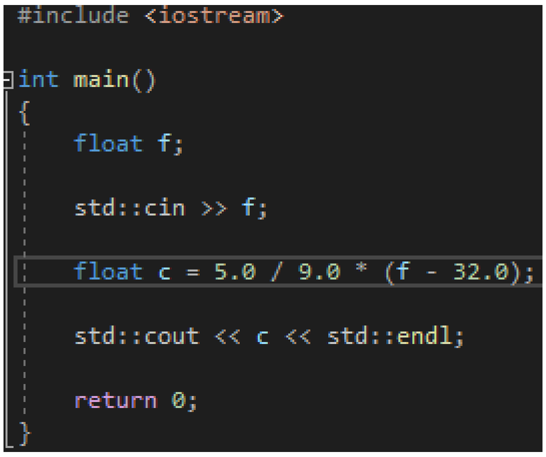
n/a

Q6. Temperature

Question:

Create a new cpp file within the temperature project and write a program to input a Fahrenheit measurement, convert it and output a Celsius value.

Solution:



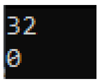
```
#include <iostream>

int main()
{
    float f;

    std::cin >> f;

    float c = 5.0 / 9.0 * (f - 32.0);

    std::cout << c << std::endl;

    return 0;
}
```

Test data:

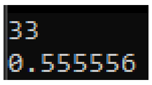
Input: 32 output: 0

Input: 33 output: 0.555556

Sample output:



```
32
0
```



```
33
0.555556
```

Reflection:

Need to remember that is I want a float answer then the numbers in the calculation should be floats as well.

Metadata:

Conversion

Further information:

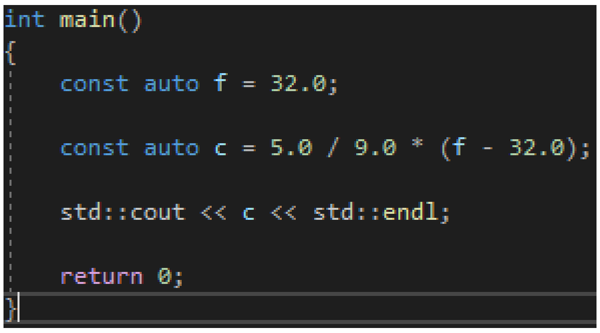
n/a

Q7. Auto, const and casting

Question:

Now rewrite your temperature example using the auto keyword, constants and explicit casting

Solution:

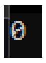


```
int main()
{
    const auto f = 32.0;

    const auto c = 5.0 / 9.0 * (f - 32.0);

    std::cout << c << std::endl;

    return 0;
}
```

Test data:

n/a

Sample output:



```
0
```

Reflection:

I would say that this takes more thought to read than the previous implementation but this defiantly uses less lines of code, but also has less functionality.

Metadata:

keywords

Further information:

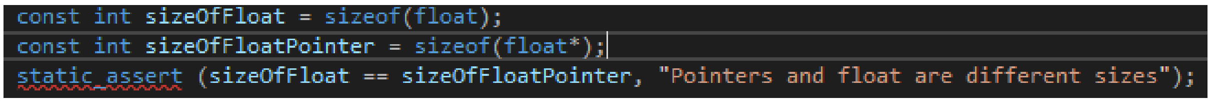
What would be some use cases for the auto keyword?

Q8. Static assert

Question:

Create a new project call sizeOf. Select a different architecture (e.g., x86 or x64) to see if you can make the assert fail.

Solution:



```
const int sizeOfFloat = sizeof(float);
const int sizeOfFloatPointer = sizeof(float*);
static_assert (sizeOfFloat == sizeOfFloatPointer, "Pointers and float are different sizes");
```

Test data:

Switch between x86 and x64 architecture

Sample output:

n/a

Reflection:

The assert doesn't work on the x64 architecture. I'm guessing we will find out why when the lectures start.

Metadata:

asserts

Further information:

n/a

Week 2 – Lab B

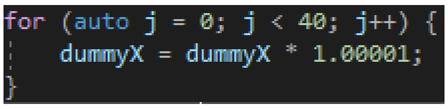
Date: 04/10/2022

Q1. Timing

Question:

Run the given application on your pc and examine the results. Try increasing the loop limit and architecture.

Solution:



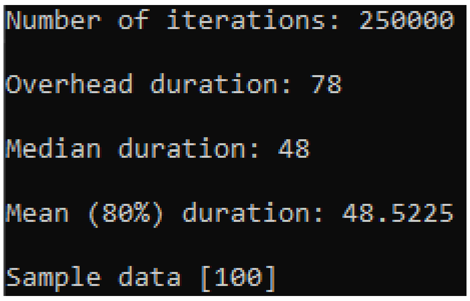
```
for (auto j = 0; j < 40; j++) {
    dummyX = dummyX * 1.00001;
}
```

Test data:

Loop limit = 20 Loop limit = 40

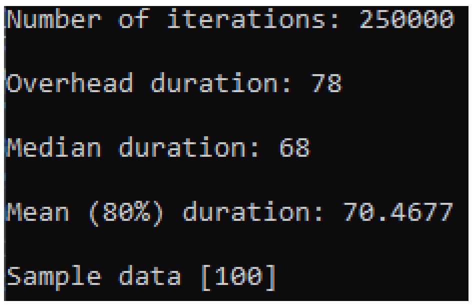
Sample output:

Loop limit: 20   Architecture: x64



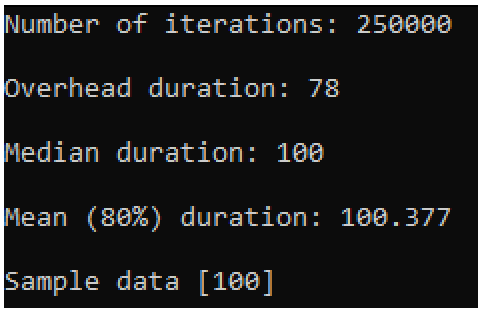
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 48

Mean (80%) duration: 48.5225

Sample data [100]
```

Loop limit: 20 Architecture: x86



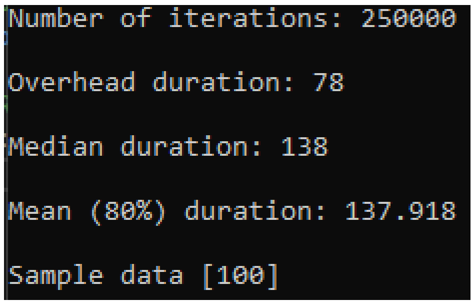
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 68

Mean (80%) duration: 70.4677

Sample data [100]
```

Loop limit: 40   Architecture: x64



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 100

Mean (80%) duration: 100.377

Sample data [100]
```

Loop limit: 40   Architecture: x86



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 138

Mean (80%) duration: 137.918

Sample data [100]
```

Reflection:

There is an increase in median duration when on a x86 architecture.

When increasing the loop limit the mean duration is a little over double the time. The only explanation I can think of is since there is double the number of loops then there is more work to do and more opportunity of the branching to be wrong.

Metadata:

Timings

Further information:

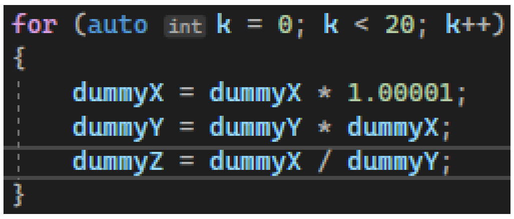
Remember that timing in debug is pointless

Q2. Timing own code

Question:

Replace the payload with some of your own code

Solution:



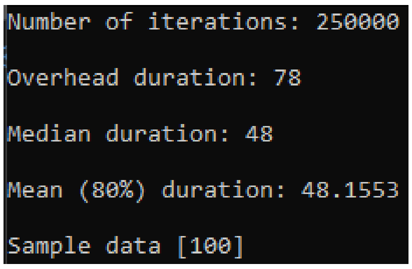
```
for (auto int k = 0; k < 20; k++)
{
    dummyX = dummyX * 1.00001;
    dummyY = dummyY * dummyX;
    dummyZ = dummyX / dummyY;
}
```

Test data:

Loop limit = 20 Loop limit = 40

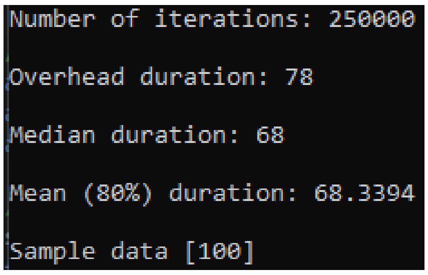
Sample output:

Target: 20   Architecture: x64



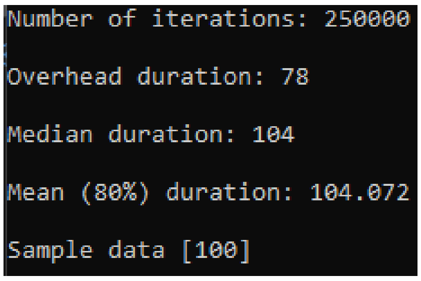
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 48

Mean (80%) duration: 48.1553

Sample data [100]
```

Target: 20 Architecture: x86



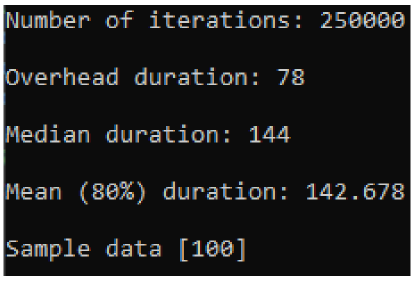
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 68

Mean (80%) duration: 68.3394

Sample data [100]
```

Target: 40   Architecture: x64



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 104

Mean (80%) duration: 104.072

Sample data [100]
```

Target: 40 Architecture: x86



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 144

Mean (80%) duration: 142.678

Sample data [100]
```

Reflection:

My timing code didn't seem to have much different times from the given example even though more operations are happening in my for loop.

Metadata:

Own code

Further information:

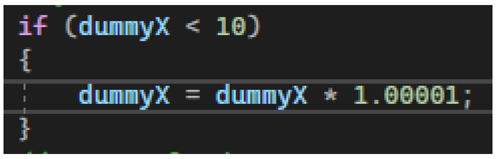
n/a

Q3. Conditionals

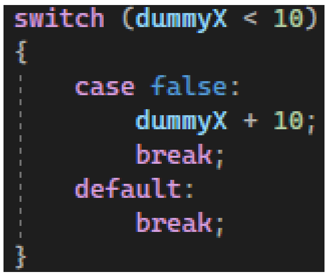
Question:

Add each of the conditional statements in turn to the payload to try and identify any performance differences.

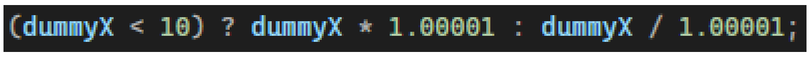
Solution:



```
if (dummyX < 10)
{
    dummyX = dummyX * 1.00001;
}
```



```
switch (dummyX < 10)
{
    case false:
        dummyX + 10;
        break;
    default:
        break;
}
```



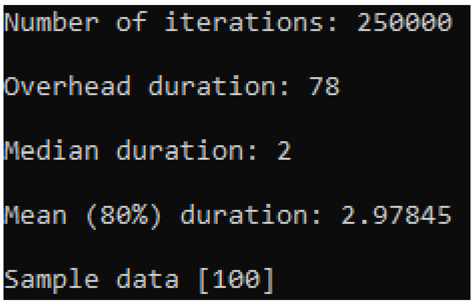
```
(dummyX < 10) ? dummyX * 1.00001 : dummyX / 1.00001;
```

Test data:

n/a

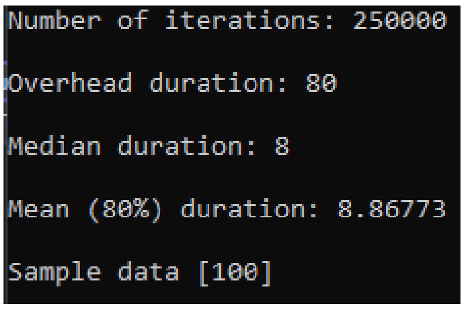
Sample output:

If x64 –



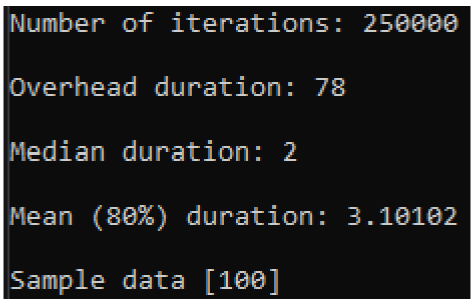
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 2

Mean (80%) duration: 2.97845

Sample data [100]
```

If x86 –



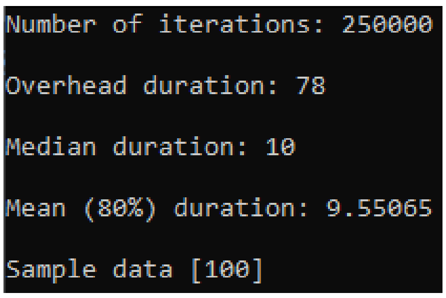
```
Number of iterations: 250000

Overhead duration: 80

Median duration: 8

Mean (80%) duration: 8.86773

Sample data [100]
```

Switch x64 –



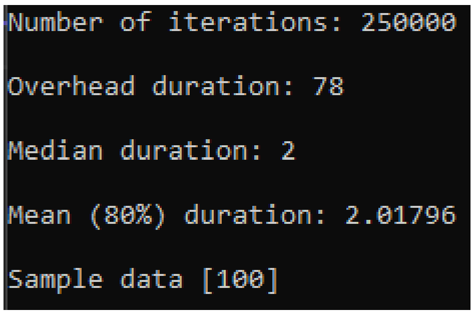
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 2

Mean (80%) duration: 3.10102

Sample data [100]
```

Switch x86 –

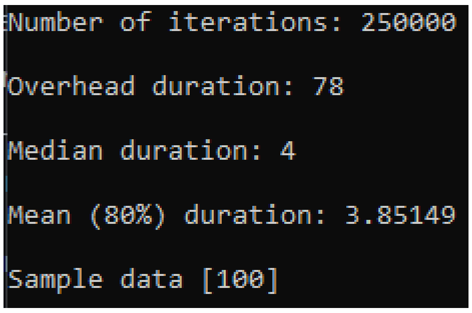


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 10

Mean (80%) duration: 9.55065

Sample data [100]
```

?: x64 –


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 2

Mean (80%) duration: 2.01796

Sample data [100]
```

?: x86 –


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 4

Mean (80%) duration: 3.85149

Sample data [100]
```

Reflection:

All of the different conditionals seem pretty similar, they all do worse on x86 architecture as expected. It seems the ?: does a little bit better than if on a x86 architecture though

Metadata:

Conditions

Further information:

n/a

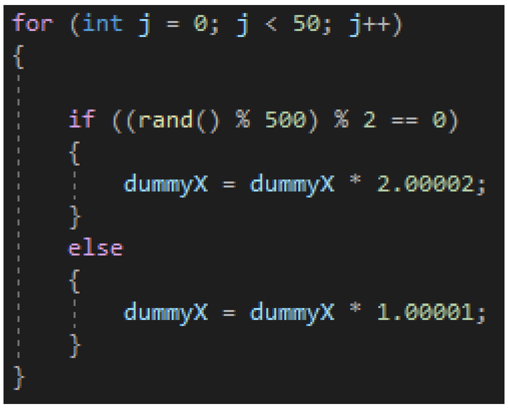
Q4. Branch prediction

Question:

Add a piece of code to the payload that demonstrates when branch prediction is working well and when branch prediction is failing

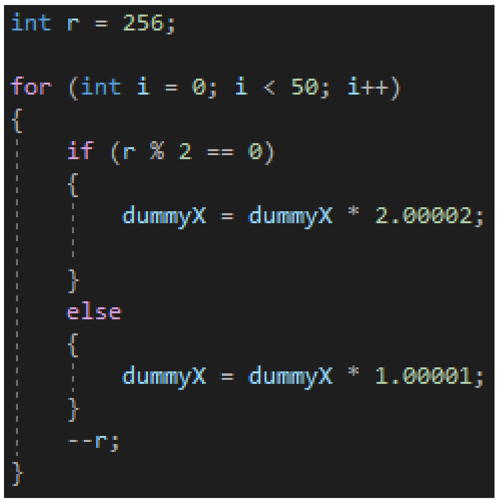
Solution:

Fail



```
for (int j = 0; j < 50; j++)
{

    if ((rand() % 500) % 2 == 0)
    {
        dummyX = dummyX * 2.00002;
    }
    else
    {
        dummyX = dummyX * 1.00001;
    }
}
```

Pass



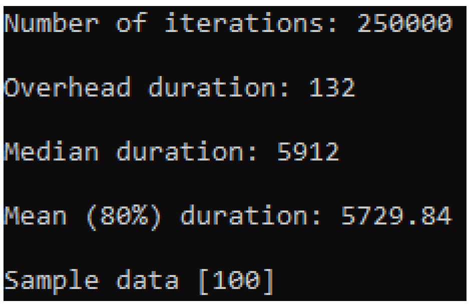
```
int r = 256;

for (int i = 0; i < 50; i++)
{
    if (r % 2 == 0)
    {
        dummyX = dummyX * 2.00002;

    }
    else
    {
        dummyX = dummyX * 1.00001;
    }
    --r;
}
```

Test data:

n/a

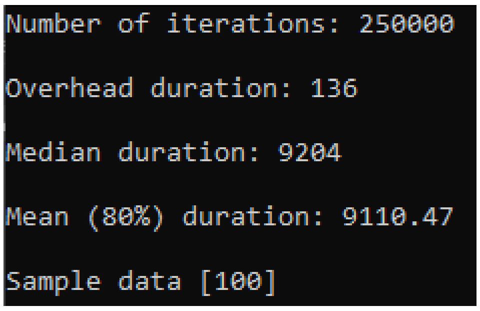
Sample output:

Fail x64



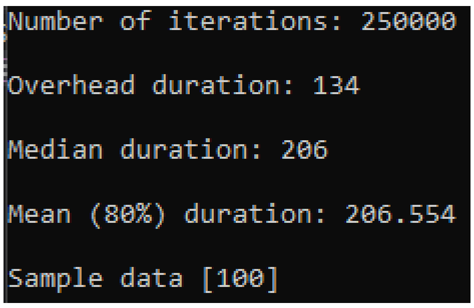
```
Number of iterations: 250000

Overhead duration: 132

Median duration: 5912

Mean (80%) duration: 5729.84

Sample data [100]
```

x86



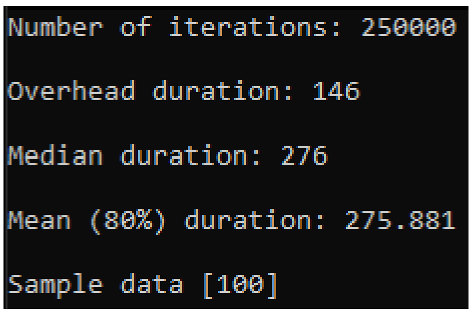
```
Number of iterations: 250000

Overhead duration: 136

Median duration: 9204

Mean (80%) duration: 9110.47

Sample data [100]
```

Pass x64



```
Number of iterations: 250000

Overhead duration: 134

Median duration: 206

Mean (80%) duration: 206.554

Sample data [100]
```

x86



```
Number of iterations: 250000

Overhead duration: 146

Median duration: 276

Mean (80%) duration: 275.881

Sample data [100]
```

Reflection:

I am not 100% sure that I have done this right but the "fail" example defiantly does worse in terms for speed. The branch predictor should only have a 50-50 chance on predicting right in the fail example because the conditions are reliant on a random number.

Metadata:

Failure

Further information:

Check with Warren to see if I have done this correctly

Q5. Exiting a nested loop

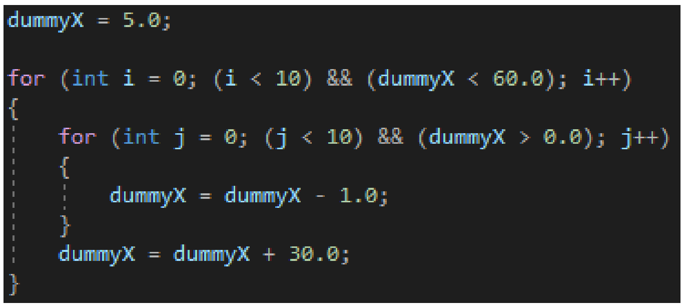
Question:

Add each option for exiting a nested loop to the payload section of the timing code and determine if there are any performance differences between each approach

1. Two conditions in each conditional section of the loops. One for the loop control and the other as the exit condition
2. An additional if statement immediately following the inner loop to catch and propagate a break statement
3. A goto statement in the inner loop
4. A lambda function

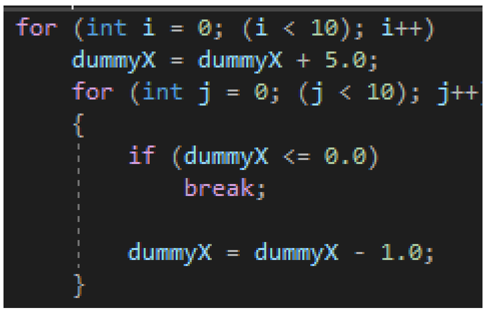
Solution:

1.



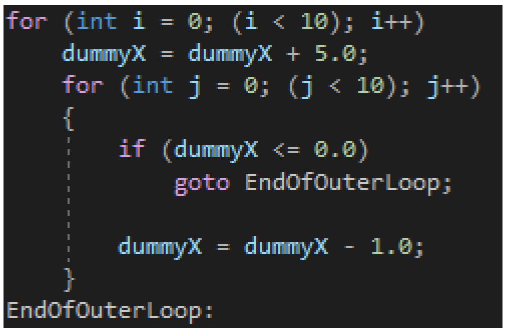
```
dummyX = 5.0;

for (int i = 0; (i < 10) && (dummyX < 60.0); i++)
{
    for (int j = 0; (j < 10) && (dummyX > 0.0); j++)
    {
        dummyX = dummyX - 1.0;
    }
    dummyX = dummyX + 30.0;
}
```

2.



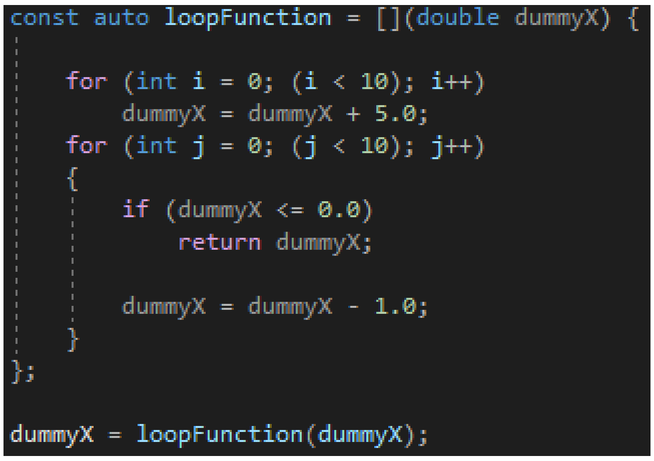
```
for (int i = 0; (i < 10); i++)
    dummyX = dummyX + 5.0;
    for (int j = 0; (j < 10); j++
    {
        if (dummyX <= 0.0)
            break;

        dummyX = dummyX - 1.0;
    }
```

3.



```
for (int i = 0; (i < 10); i++)
    dummyX = dummyX + 5.0;
    for (int j = 0; (j < 10); j++)
    {
        if (dummyX <= 0.0)
            goto EndOfOuterLoop;

        dummyX = dummyX - 1.0;
    }
EndOfOuterLoop:
```

4.



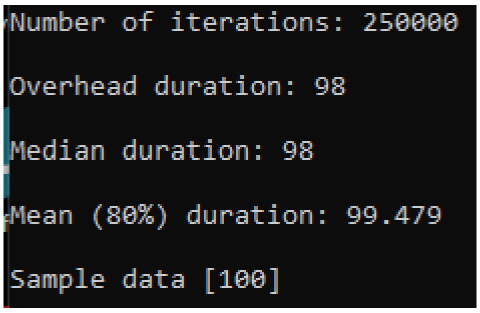
```
const auto loopFunction = [](double dummyX) {

    for (int i = 0; (i < 10); i++)
        dummyX = dummyX + 5.0;
    for (int j = 0; (j < 10); j++)
    {
        if (dummyX <= 0.0)
            return dummyX;

        dummyX = dummyX - 1.0;
    }
};

dummyX = loopFunction(dummyX);
```

Test data:

n/a

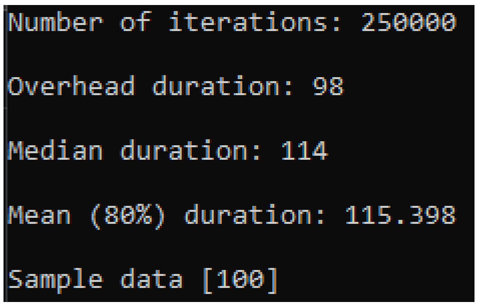
Sample output:

1. x64



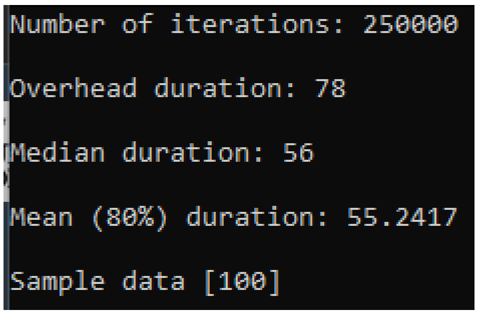
```
Number of iterations: 250000

Overhead duration: 98

Median duration: 98

Mean (80%) duration: 99.479

Sample data [100]
```

x86



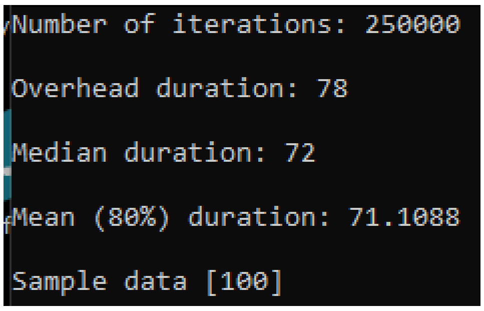
```
Number of iterations: 250000

Overhead duration: 98

Median duration: 114

Mean (80%) duration: 115.398

Sample data [100]
```

2. x64



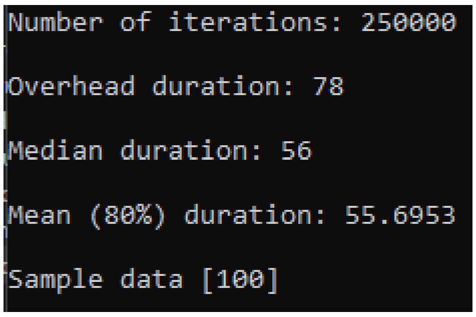
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 56

Mean (80%) duration: 55.2417

Sample data [100]
```

x86

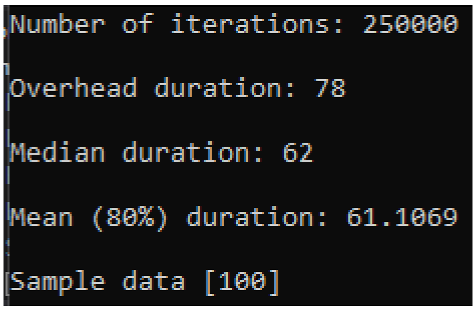


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 72

Mean (80%) duration: 71.1088

Sample data [100]
```

3. x64 x86

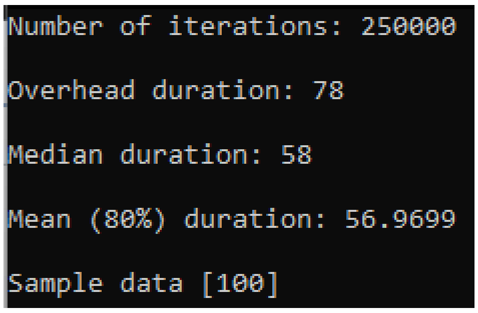


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 56

Mean (80%) duration: 55.6953

Sample data [100]
```

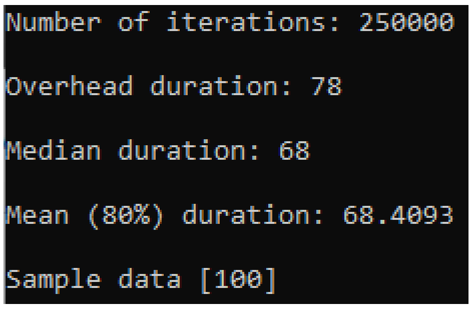


```
Number of iterations: 250000

Overhead duration: 78

Median duration: 62

Mean (80%) duration: 61.1069

Sample data [100]
```

4. x64 x86



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 58

Mean (80%) duration: 56.9699

Sample data [100]
```



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 68

Mean (80%) duration: 68.4093

Sample data [100]
```

Reflection:

I am not sure I have implemented all of these correctly but based on my results all of the methods of exiting a loop are fairly similar in speed except for method 1 (Two conditions in each conditional section of the loops. One for the loop control and the other as the exit condition). When running the tests in x86 architecture they all run slower than the x64 as expected.

Metadata:

Nested

Further information:

They are supposed to be all pretty equal method 1 is wrong because of the overhead

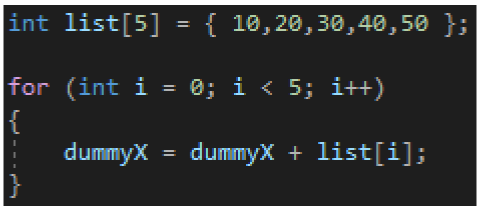
Q6. Range based loop

Question:

Compare the performance of standard loops and range based loops

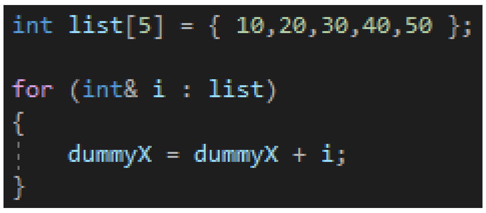
Solution:

Standard



```
int list[5] = { 10,20,30,40,50 };

for (int i = 0; i < 5; i++)
{
    dummyX = dummyX + list[i];
}
```

Range-Based



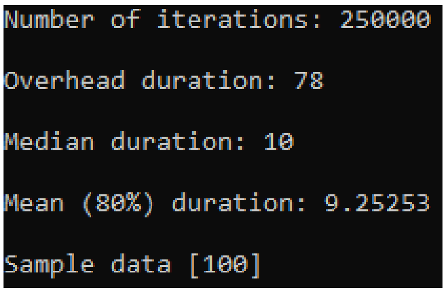
```
int list[5] = { 10,20,30,40,50 };

for (int& i : list)
{
    dummyX = dummyX + i;
}
```

Test data:

n/a

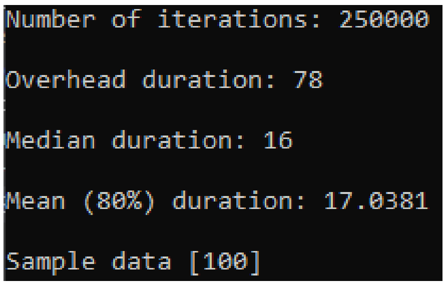
Sample output:

Standard x64



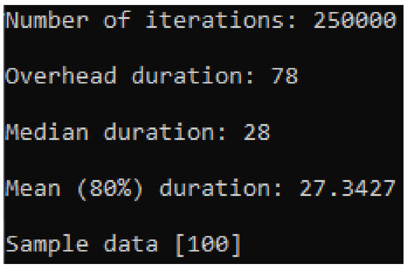
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 10

Mean (80%) duration: 9.25253

Sample data [100]
```

x86



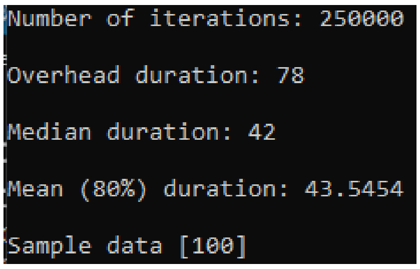
```
Number of iterations: 250000

Overhead duration: 78

Median duration: 16

Mean (80%) duration: 17.0381

Sample data [100]
```

Range-Based x64



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 28

Mean (80%) duration: 27.3427

Sample data [100]
```

x86



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 42

Mean (80%) duration: 43.5454

Sample data [100]
```

Reflection:

Ranged-based is significantly slower both in x64 and x86. I was expecting it to be quicker. Maybe I've implemented it wrong

Metadata:

Range-Based

Further information:

Are my results as expected? If so why is it so much slower? **They are actually pretty equal in speed**

Q7. Architecture

Question:

Run the previous experiments in both x64 and x86 architecture

Solution:

n/a

Test data:

n/a

Sample output:

n/a

Reflection:

I’ve done this as part of each question and commented on the architecture differences in each

Metadata:

X86

Further information:

n/a

Week 3 – Lab C

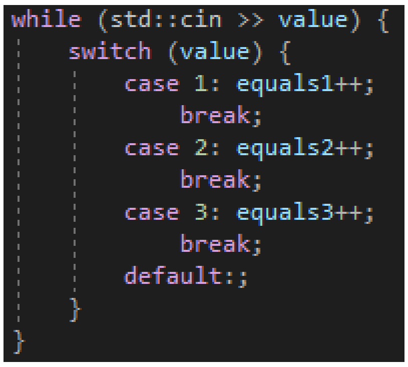
Date: 11/10/2022

Q1. Debugging

Question:

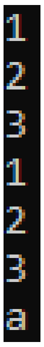
Determine what the problem is with the program and suggest a solution to make the program execute correctly.

Solution:

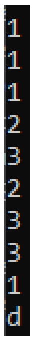


```
while (std::cin >> value) {
    switch (value) {
        case 1: equals1++;
            break;
        case 2: equals2++;
            break;
        case 3: equals3++;
            break;
        default:;
    }
}
```

Test data:

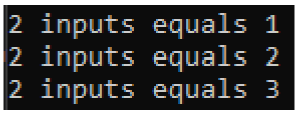


```
1
2
3
1
2
3
a
```

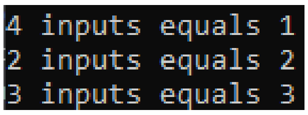


```
1
1
1
2
3
2
3
3
1
d
```

Sample output:



```
2 inputs equals 1
2 inputs equals 2
2 inputs equals 3
```



```
4 inputs equals 1
2 inputs equals 2
3 inputs equals 3
```

Reflection:

The problem was that the switch didn't have a break at the end of each case causing the code to fall through. So, if case 1 was triggered so would case 2 and 3. The fix is simply adding in the breaks for each case

Metadata:

Switch error

Further information:

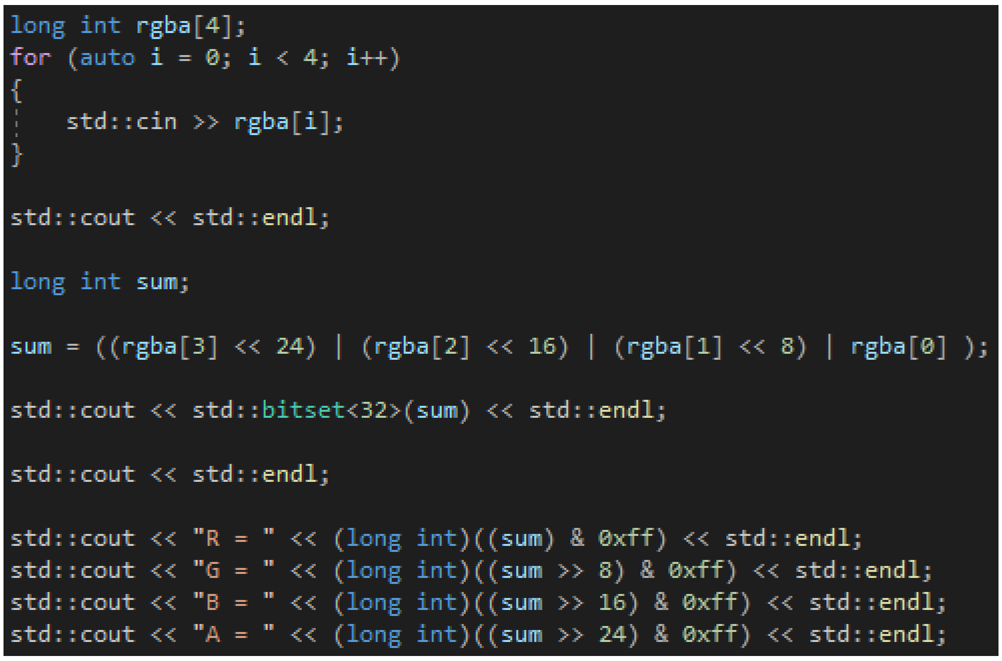
n/a

Q2. Bitwise

Question:

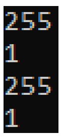
Write a program to read four separate 32-bit integers (red, green, blue, and alpha) and encode them into a single 32-bit value. Output this 32-bit value. Verify that the results are correct by taking the 32-bit value and extracting and outputting the separate integers.

Solution:

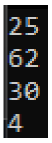


```
long int rgba[4];
for (auto i = 0; i < 4; i++)
{
    std::cin >> rgba[i];
}

std::cout << std::endl;

long int sum;

sum = ((rgba[3] << 24) | (rgba[2] << 16) | (rgba[1] << 8) | rgba[0] );

std::cout << std::bitset<32>(sum) << std::endl;

std::cout << std::endl;

std::cout << "R = " << (long int)((sum) & 0xff) << std::endl;
std::cout << "G = " << (long int)((sum >> 8) & 0xff) << std::endl;
std::cout << "B = " << (long int)((sum >> 16) & 0xff) << std::endl;
std::cout << "A = " << (long int)((sum >> 24) & 0xff) << std::endl;
```

Test data:

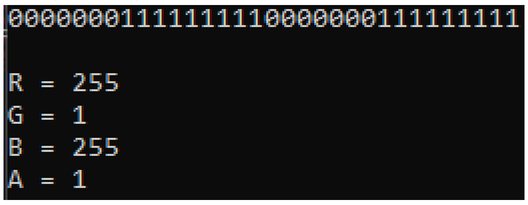


```
255
1
255
1
```

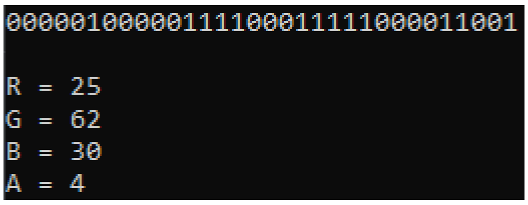


```
25
62
30
4
```

Sample output:



```
00000001111111110000000111111111

R = 255
G = 1
B = 255
A = 1
```



```
00000100000111100011111000011001

R = 25
G = 62
B = 30
A = 4
```

Reflection:

This took a while and I had to get quite a bit of help but I think I understand bitwise a bit better now. I should still do some more research into bitwise stuff though.

Metadata:

Bitwise encoding

Further information:

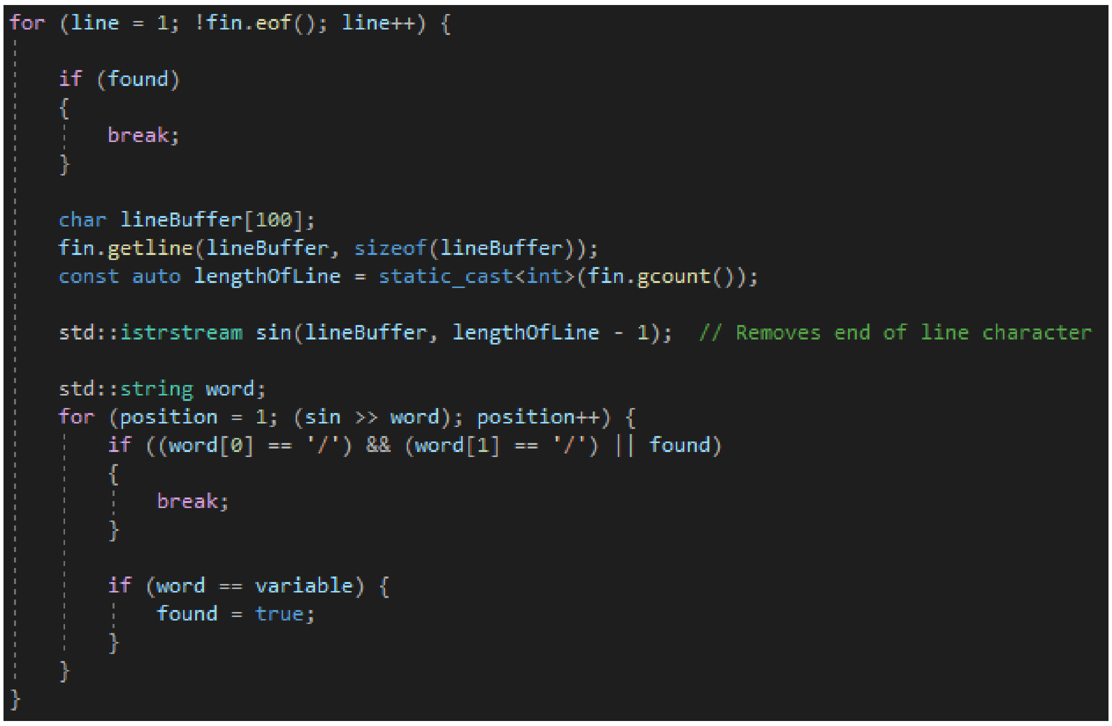
n/a

Q3. Parsing

Question:

Modify the program to make the loop structures more efficient and easier to maintain. Time the code to determine if you have made to code more efficient.

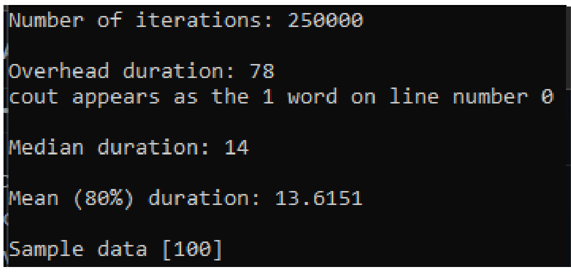
Solution:



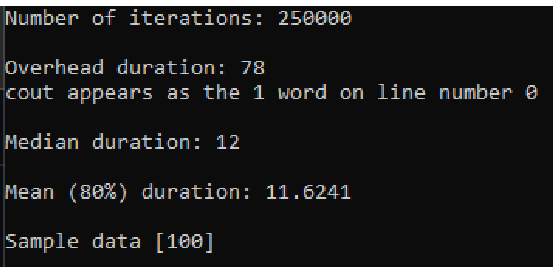
```
for (line = 1; !fin.eof(); line++) {

    if (found)
    {
        break;
    }

    char lineBuffer[100];
    fin.getline(lineBuffer, sizeof(lineBuffer));
    const auto lengthOfLine = static_cast<int>(fin.gcount());

    std::istrstream sin(lineBuffer, lengthOfLine - 1);  // Removes end of line character

    std::string word;
    for (position = 1; (sin >> word); position++) {
        if ((word[0] == '/') && (word[1] == '/') || found)
        {
            break;
        }

        if (word == variable) {
            found = true;
        }
    }
}
```

Test data:

Release x64 Original   Mine

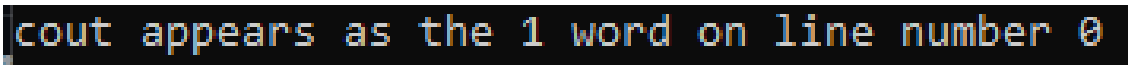


```
Number of iterations: 250000

Overhead duration: 78
cout appears as the 1 word on line number 0

Median duration: 14

Mean (80%) duration: 13.6151

Sample data [100]
```



```
Number of iterations: 250000

Overhead duration: 78
cout appears as the 1 word on line number 0

Median duration: 12

Mean (80%) duration: 11.6241

Sample data [100]
```

Sample output:



```
cout appears as the 1 word on line number 0
```

Reflection:

My solution is ever so slightly faster however when I was doing the timing the times for the original seemed to vary wildly so I'm not 100% sure if it is more efficient. In terms of maintainability, I'd say the 2 solutions are equal and I'm not sure how you could improve maintainability.

Metadata:

Parsing improvement

Further information:

n/a

Q4. Quadratic

Question:

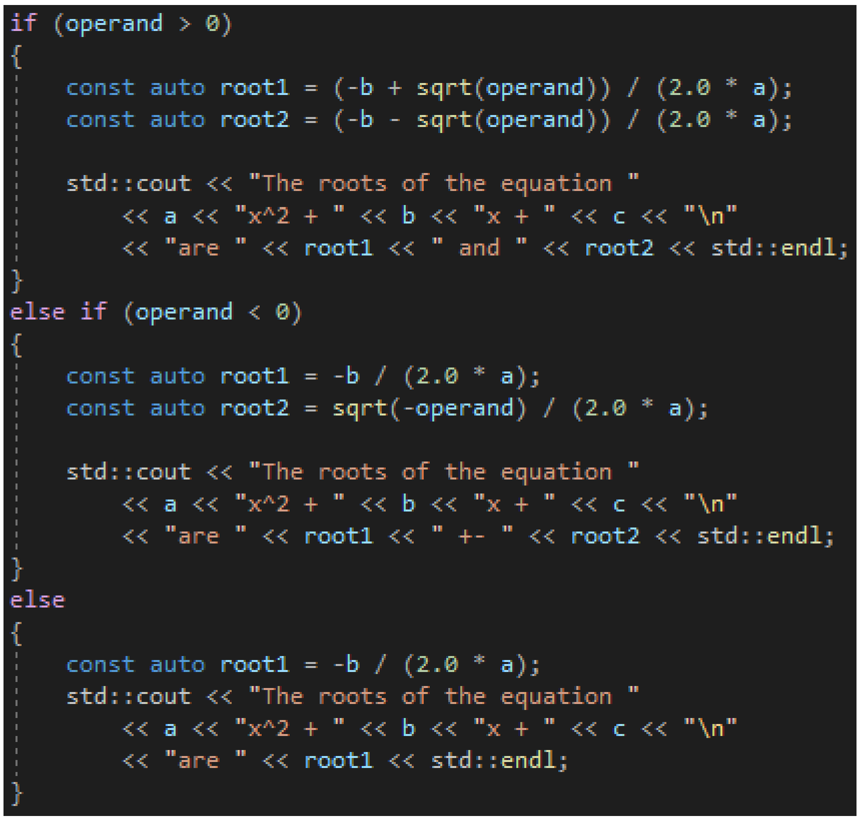
Modify the quadratic program to:

1. Handle equal or imaginary roots
2. Output the roots to only 3 decimal places

Time the new code and try to improve efficiency

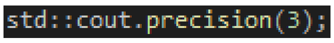
Solution:

1.



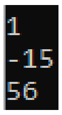
```cpp
if (operand > 0)
{
    const auto root1 = (-b + sqrt(operand)) / (2.0 * a);
    const auto root2 = (-b - sqrt(operand)) / (2.0 * a);

    std::cout << "The roots of the equation "
        << a << "x^2 + " << b << "x + " << c << "\n"
        << "are " << root1 << " and " << root2 << std::endl;
}
else if (operand < 0)
{
    const auto root1 = -b / (2.0 * a);
    const auto root2 = sqrt(-operand) / (2.0 * a);

    std::cout << "The roots of the equation "
        << a << "x^2 + " << b << "x + " << c << "\n"
        << "are " << root1 << " +- " << root2 << std::endl;
}
else
{
    const auto root1 = -b / (2.0 * a);
    std::cout << "The roots of the equation "
        << a << "x^2 + " << b << "x + " << c << "\n"
        << "are " << root1 << std::endl;
}
```

2.

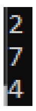


```cpp
std::cout.precision(3);
```

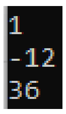
Test data:



```
1
-15
56
```

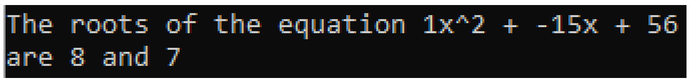


```
2
7
4
```

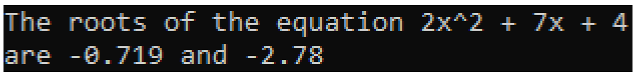


```
1
-12
36
```

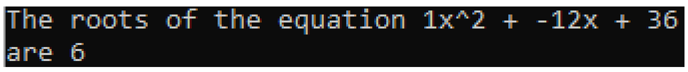
Sample output:



```
The roots of the equation 1x^2 + -15x + 56
are 8 and 7
```



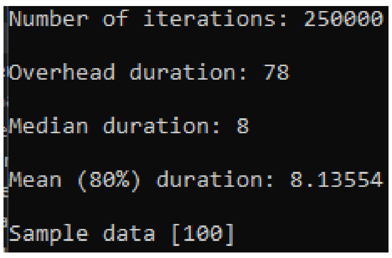
```
The roots of the equation 2x^2 + 7x + 4
are -0.719 and -2.78
```



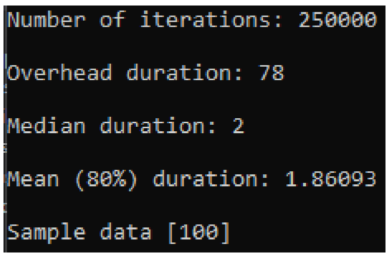
```
The roots of the equation 1x^2 + -12x + 36
are 6
```

Release x64

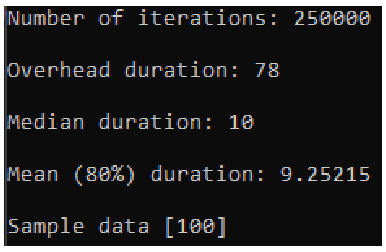
Real & Diff



```
Number of iterations: 250000
Overhead duration: 78
Median duration: 8
Mean (80%) duration: 8.13554
Sample data [100]
```

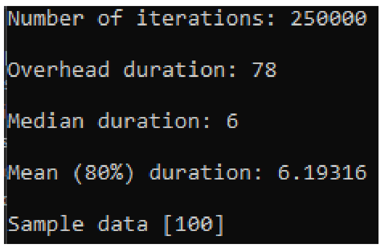


```
Number of iterations: 250000
Overhead duration: 78
Median duration: 2
Mean (80%) duration: 1.86093
Sample data [100]
```

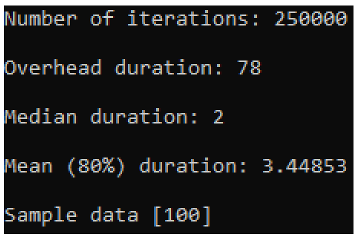


```
Number of iterations: 250000
Overhead duration: 78
Median duration: 10
Mean (80%) duration: 9.25215
Sample data [100]
```

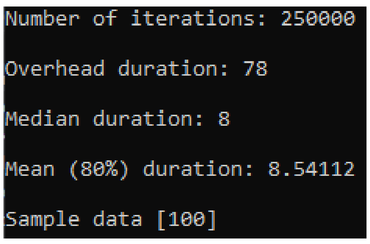
Imaginary



```
Number of iterations: 250000
Overhead duration: 78
Median duration: 6
Mean (80%) duration: 6.19316
Sample data [100]
```

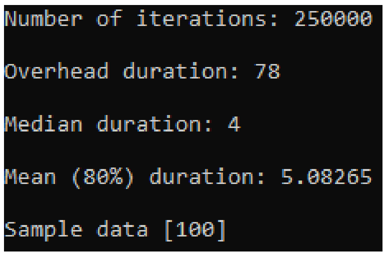


```
Number of iterations: 250000
Overhead duration: 78
Median duration: 2
Mean (80%) duration: 3.44853
Sample data [100]
```

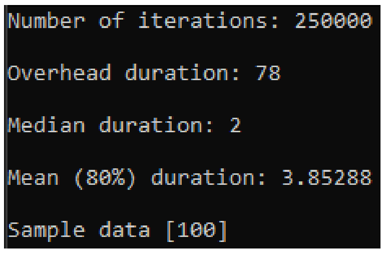


```
Number of iterations: 250000
Overhead duration: 78
Median duration: 8
Mean (80%) duration: 8.54112
Sample data [100]
```

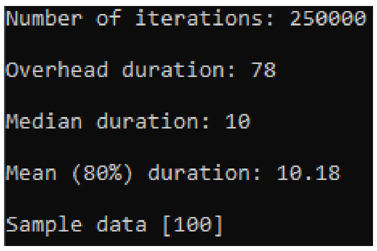
Real Same



```
Number of iterations: 250000
Overhead duration: 78
Median duration: 4
Mean (80%) duration: 5.08265
Sample data [100]
```



```
Number of iterations: 250000
Overhead duration: 78
Median duration: 2
Mean (80%) duration: 3.85288
Sample data [100]
```



```
Number of iterations: 250000
Overhead duration: 78
Median duration: 10
Mean (80%) duration: 10.18
Sample data [100]
```

Reflection:

This was really hard to do purely because I don't remember anything about quadratic equations. When I came to the try to improve efficiency part I couldn't think of any way to get any improvement. I need to use a conditional for the program and the other types of conditionals aren't appropriate. I think I remember seeing Warren do the precision differently in a lecture but I couldn't find it or remember it, I think % was used somehow.

Metadata:

Quadratic equations

Further information:

I think I saw Warren do precision another way so I need to ask him how he does it

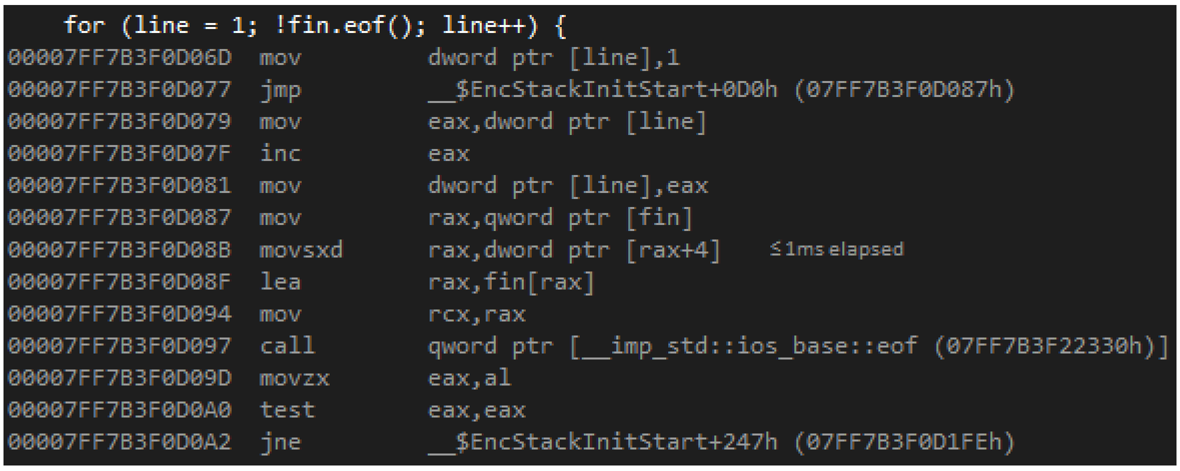
I noticed that the overhead massively increases if my laptop is unplugged. It goes from ~80 when plugged in to ~200 when unplugged.

Q5. Assembly

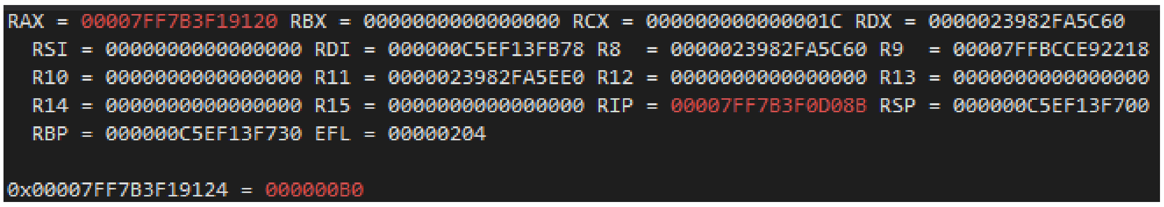
Question:

Open and step through the assembly code from the parsing exercise. Do this in debug mode.

Solution:



```
    for (line = 1; !fin.eof(); line++) {
00007FF7B3F0D06D  mov         dword ptr [line],1
00007FF7B3F0D077  jmp         __$EncStackInitStart+0D0h (07FF7B3F0D087h)
00007FF7B3F0D079  mov         eax,dword ptr [line]
00007FF7B3F0D07F  inc         eax
00007FF7B3F0D081  mov         dword ptr [line],eax
00007FF7B3F0D087  mov         rax,qword ptr [fin]
00007FF7B3F0D08B  movsxd      rax,dword ptr [rax+4]    ≤1ms elapsed
00007FF7B3F0D08F  lea         rax,fin[rax]
00007FF7B3F0D094  mov         rcx,rax
00007FF7B3F0D097  call        qword ptr [__imp_std::ios_base::eof (07FF7B3F22330h)]
00007FF7B3F0D09D  movzx       eax,al
00007FF7B3F0D0A0  test        eax,eax
00007FF7B3F0D0A2  jne         __$EncStackInitStart+247h (07FF7B3F0D1FEh)
```



```
RAX = 00007FF7B3F19120 RBX = 0000000000000000 RCX = 000000000000001C RDX = 0000023982FA5C60
  RSI = 0000000000000000 RDI = 000000C5EF13FB78 R8  = 0000023982FA5C60 R9  = 00007FFBCCE92218
  R10 = 0000000000000000 R11 = 0000023982FA5EE0 R12 = 0000000000000000 R13 = 0000000000000000
  R14 = 0000000000000000 R15 = 0000000000000000 RIP = 00007FF7B3F0D08B RSP = 000000C5EF13F700
  RBP = 000000C5EF13F730 EFL = 00000204

0x00007FF7B3F19124 = 000000B0
```

Test data:

n/a

Sample output:

n/a

Reflection:

I've gone through the assembly and I understand what is happening for the most part but I think that's only due to the fact that I know what's supposed to be happening in the code, if I was just given the assembly code with no other context I don't think I'd be able to tell you what the related piece of code was doing.

After going through the assembly code, I noticed there were a few instructions that I didn't recognise so I've found out what they mean and put them below:

nop = No operation

inc = Increment (+1)

movsx/movsxd = Move with Sign-Extension

movzx = Move with Zero-Extend

test = Preforms a bitwise and

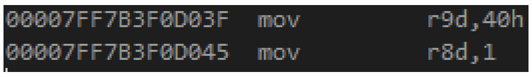
dec = Decrement (-1)

xor = Perform a bitwise exclusive or

Metadata:

Parsing Assembly

Further information:



```
00007FF7B3F0D03F  mov          r9d,40h
00007FF7B3F0D045  mov          r8d,1
```

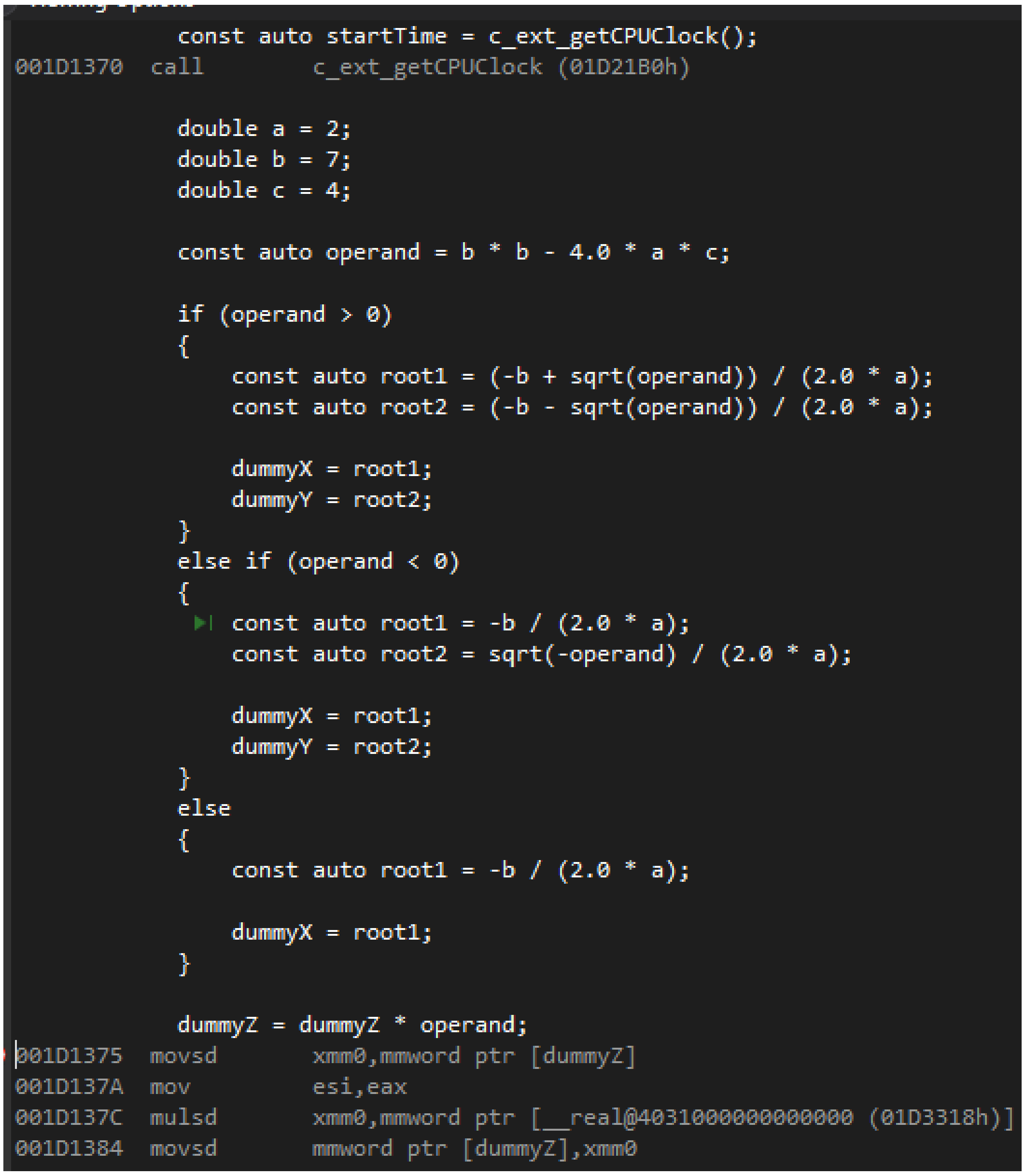
Why is there a d on the end of the R9 and R8 register?

Q6. Optimizer

Question:

In release mode disassemble the code you wrote for the quadratic exercise in the timing code.

Solution:



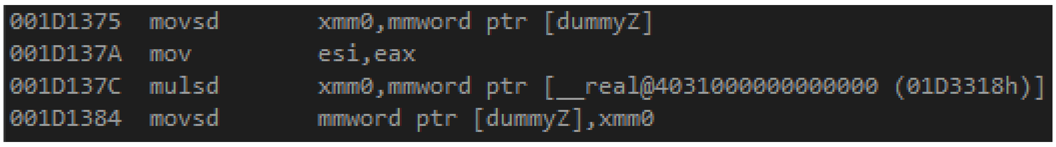
```
            const auto startTime = c_ext_getCPUClock();
001D1370  call        c_ext_getCPUClock (01D21B0h)

            double a = 2;
            double b = 7;
            double c = 4;

            const auto operand = b * b - 4.0 * a * c;

            if (operand > 0)
            {
                const auto root1 = (-b + sqrt(operand)) / (2.0 * a);
                const auto root2 = (-b - sqrt(operand)) / (2.0 * a);

                dummyX = root1;
                dummyY = root2;
            }
            else if (operand < 0)
            {
                const auto root1 = -b / (2.0 * a);
                const auto root2 = sqrt(-operand) / (2.0 * a);

                dummyX = root1;
                dummyY = root2;
            }
            else
            {
                const auto root1 = -b / (2.0 * a);

                dummyX = root1;
            }

            dummyZ = dummyZ * operand;
001D1375  movsd       xmm0,mmword ptr [dummyZ]
001D137A  mov         esi,eax
001D137C  mulsd       xmm0,mmword ptr [__real@4031000000000000 (01D3318h)]
001D1384  movsd       mmword ptr [dummyZ],xmm0
```

Test data:

n/a

Sample output:
n/a

Reflection:
The optimiser seems to have completely thrown out my code and seems to be figuring out the answers in the following section ready to print later:



```
001D1375  movsd       xmm0,mmword ptr [dummyZ]
001D137A  mov         esi,eax
001D137C  mulsd       xmm0,mmword ptr [__real@4031000000000000 (01D3318h)]
001D1384  movsd       mmword ptr [dummyZ],xmm0
```

I don't really understand what's happening or why. I though since the roots and operand are need to alter the dummy values before they get printed that the optimiser would leave my code alone.

mulsd = Multiply Scalar Double-Precision Floating-Point Value

cmovg = conditional move if greater check the state of ZF, SF AND OF. If ZF=0 AND SF=OF then condition is satisfied, otherwise it will be skipped.

cmp = subtracts one operand from the other for comparing whether the operands are equal or not

Metadata:
Optimiser

Further information:
Is this what was expected or have I done something wrong. **The code has not been optimised out it is all below the payload the disassembler just gave up doing it line by line.**

Week 4 – Lab D

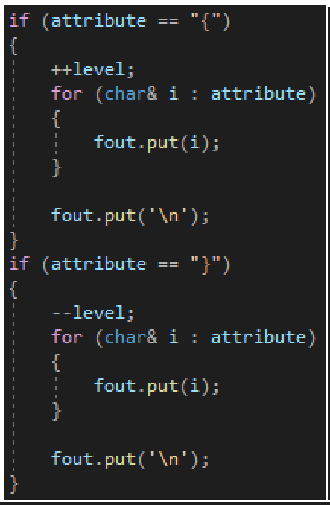
Date: 25/10/2022

Q1. Object Parser

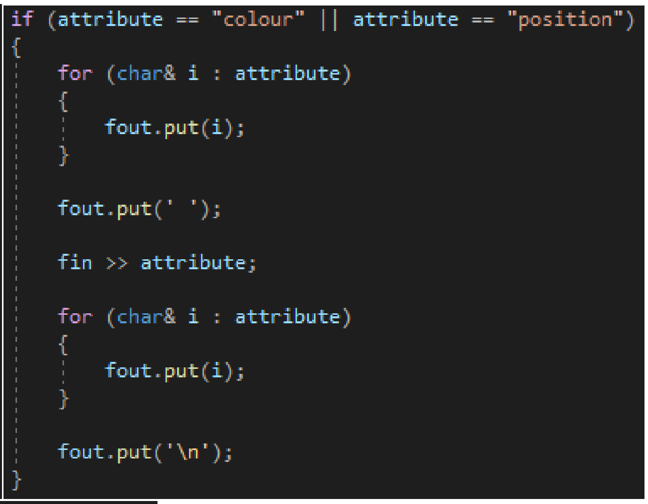
Question:

Complete the object parser code. It must be able to copy data to an output file and identify the object with the largest size.

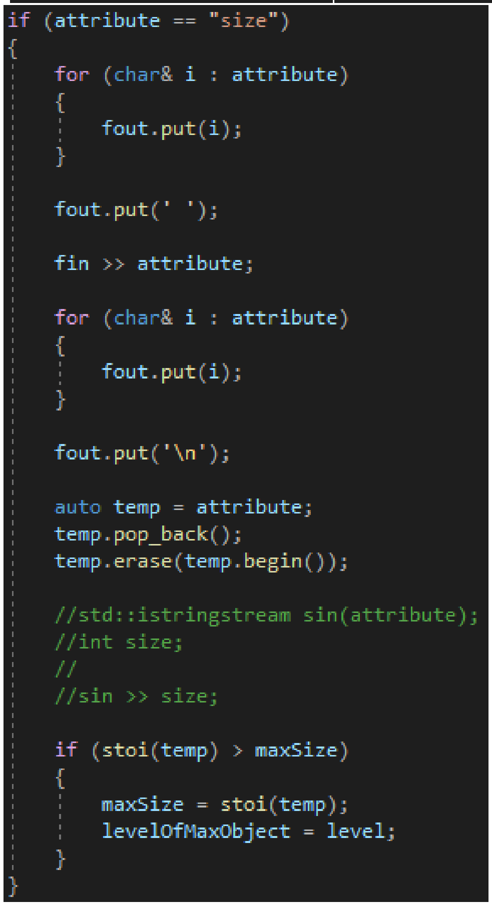
Solution:



```cpp
if (attribute == "{")
{
    ++level;
    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put('\n');
}
if (attribute == "}")
{
    --level;
    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put('\n');
}
```

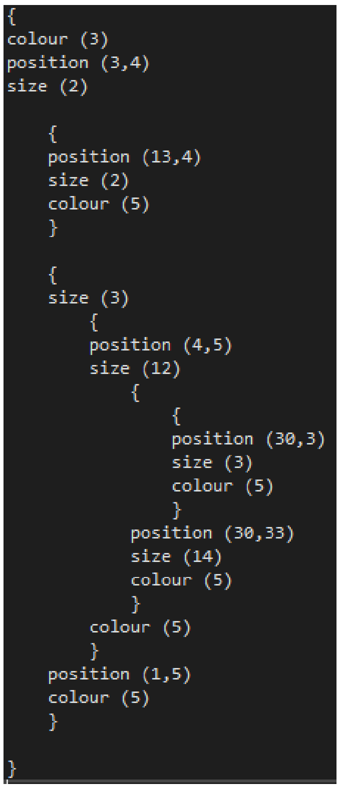


```cpp
if (attribute == "colour" || attribute == "position")
{
    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put(' ');

    fin >> attribute;

    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put('\n');
}
```



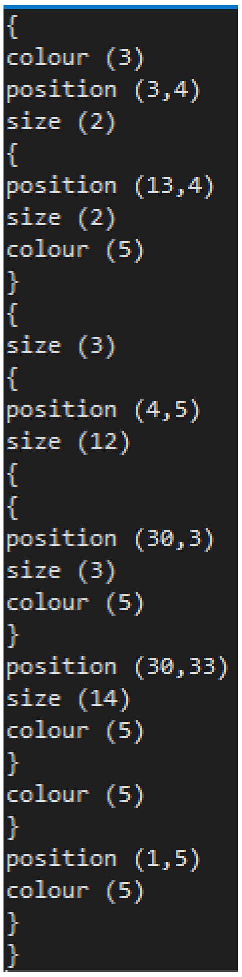
```cpp
if (attribute == "size")
{
    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put(' ');

    fin >> attribute;

    for (char& i : attribute)
    {
        fout.put(i);
    }

    fout.put('\n');

    auto temp = attribute;
    temp.pop_back();
    temp.erase(temp.begin());

    //std::istringstream sin(attribute);
    //int size;
    //
    //sin >> size;

    if (stoi(temp) > maxSize)
    {
        maxSize = stoi(temp);
        levelOfMaxObject = level;
    }
}
```

Test data:



```
{
colour (3)
position (3,4)
size (2)

    {
    position (13,4)
    size (2)
    colour (5)
    }

    {
    size (3)
        {
        position (4,5)
        size (12)
            {
                {
                position (30,3)
                size (3)
                colour (5)
                }
            position (30,33)
            size (14)
            colour (5)
            }
        colour (5)
        }
    position (1,5)
    colour (5)
    }

}
```

Sample output:



```
{
colour (3)
position (3,4)
size (2)
{
position (13,4)
size (2)
colour (5)
}
{
size (3)
{
position (4,5)
size (12)
{
{
position (30,3)
size (3)
colour (5)
}
position (30,33)
size (14)
colour (5)
}
colour (5)
}
position (1,5)
colour (5)
}
}
```

Reflection:

This wasn't too difficult the most difficult part was converting a size string into an int. Warren suggested using istringstream but I don't think I was using it right because it always gave a size of 0.

Note: I know I could have just used streaming operators for writing but I wanted to practice using range-based for loops.

Metadata:

Parsing Objects

Further information:

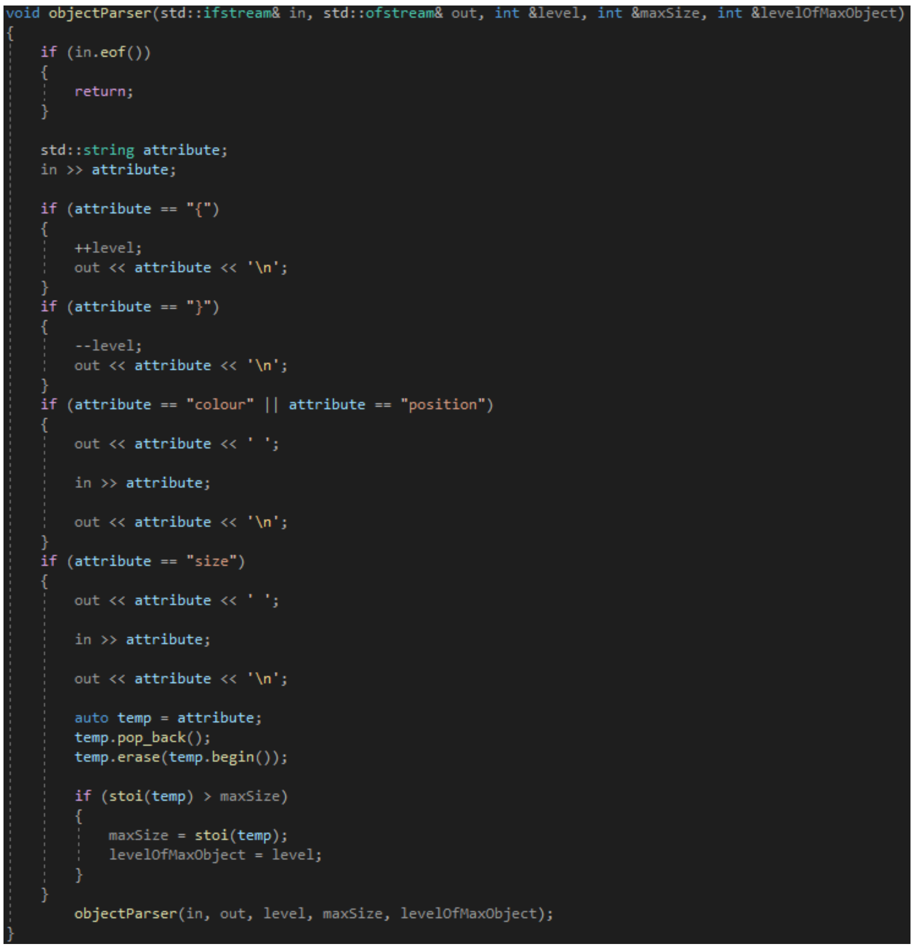
What am I doing wrong with the istringstream?

Q2. Object Parser (Recursive)

Question:

Create a recursive solution to the previous task. No loop structures allowed

Solution:



```
void objectParser(std::ifstream& in, std::ofstream& out, int &level, int &maxSize, int &levelOfMaxObject)
{
    if (in.eof())
    {
        return;
    }

    std::string attribute;
    in >> attribute;

    if (attribute == "{")
    {
        ++level;
        out << attribute << '\n';
    }
    if (attribute == "}")
    {
        --level;
        out << attribute << '\n';
    }
    if (attribute == "colour" || attribute == "position")
    {
        out << attribute << ' ';

        in >> attribute;

        out << attribute << '\n';
    }
    if (attribute == "size")
    {
        out << attribute << ' ';

        in >> attribute;

        out << attribute << '\n';

        auto temp = attribute;
        temp.pop_back();
        temp.erase(temp.begin());

        if (stoi(temp) > maxSize)
        {
            maxSize = stoi(temp);
            levelOfMaxObject = level;
        }
    }
        objectParser(in, out, level, maxSize, levelOfMaxObject);
}
```

Test data:

Same as previous task

Sample output:

Same as previous task

Reflection:

This was pretty straight forward, I probably have too many parameters so it's a bit messy and has a lot of room for errors to be made.

Metadata:

Recursion

Further information:

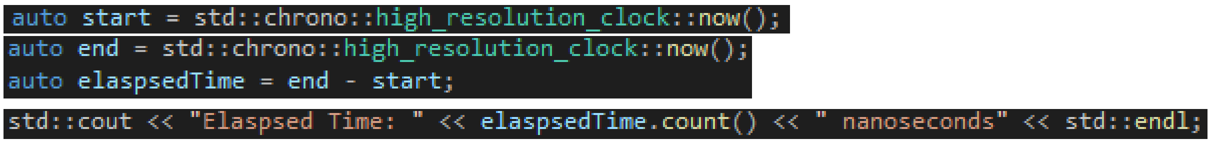
n/a

Q3. Object Parser Comparison

Question:

Write some timing code a use it to compare the speed of object parsing with loop structures and with recursion.

Solution:



```
auto start = std::chrono::high_resolution_clock::now();
auto end = std::chrono::high_resolution_clock::now();
auto elaspsedTime = end - start;
std::cout << "Elaspsed Time: " << elaspsedTime.count() << " nanoseconds" << std::endl;
```

Test data:

n/a

Sample output:

Loop structures Release x64	**Recursion Release x64**
`Elaspsed Time: 100800 nanoseconds`	`Elaspsed Time: 107600 nanoseconds`
`Elaspsed Time: 103800 nanoseconds`	`Elaspsed Time: 137400 nanoseconds`
`Elaspsed Time: 118500 nanoseconds`	`Elaspsed Time: 104800 nanoseconds`
`Elaspsed Time: 102500 nanoseconds`	`Elaspsed Time: 100400 nanoseconds`
`Elaspsed Time: 107200 nanoseconds`	`Elaspsed Time: 111300 nanoseconds`
Average Time = 106560 nanoseconds	*Average Time = 112300 nanoseconds*

Reflection:

The timings seem pretty similar loop structures do come out to be a bit faster in my timings but this is probably just luck of the draw since I just took 5 random timings for each solution. I probably should have looped through multiple iterations and gotten the average like that.

Metadata:

Comparison

Further information:

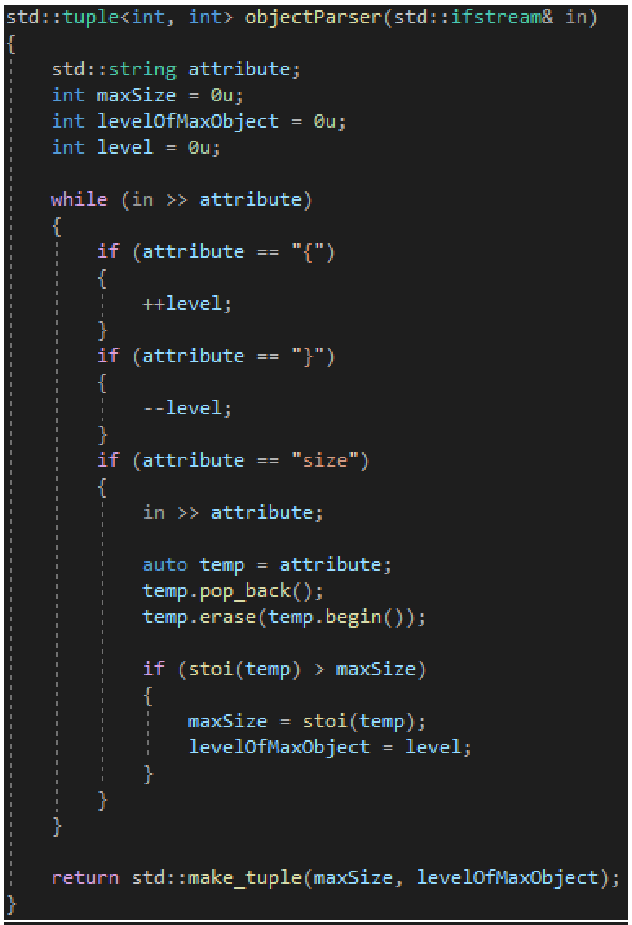
For the loop structures I changed to way things a written to the output.txt so they matched across both solutions

Q4. Tuples

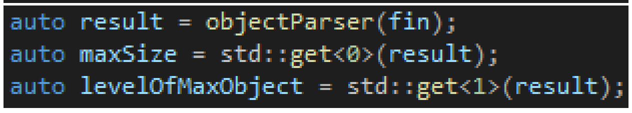
Question:

Select one of your object parsers and wrap the code within a new function. This new function has a single parameter, the file name, and returns a tuple consisting of the size of the largest object and the level of the largest object

Solution:



```cpp
std::tuple<int, int> objectParser(std::ifstream& in)
{
    std::string attribute;
    int maxSize = 0u;
    int levelOfMaxObject = 0u;
    int level = 0u;

    while (in >> attribute)
    {
        if (attribute == "{")
        {
            ++level;
        }
        if (attribute == "}")
        {
            --level;
        }
        if (attribute == "size")
        {
            in >> attribute;

            auto temp = attribute;
            temp.pop_back();
            temp.erase(temp.begin());

            if (stoi(temp) > maxSize)
            {
                maxSize = stoi(temp);
                levelOfMaxObject = level;
            }
        }
    }

    return std::make_tuple(maxSize, levelOfMaxObject);
}
```

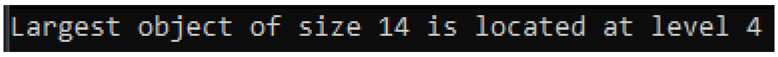


```cpp
auto result = objectParser(fin);
auto maxSize = std::get<0>(result);
auto levelOfMaxObject = std::get<1>(result);
```

Test data:

Sample.txt

Sample output:



```
Largest object of size 14 is located at level 4
```

Reflection:

This was pretty simply to do, I cut out most of the code since we aren't writing to the output file for this question.

Metadata:

Tuple

Further information:

n/a

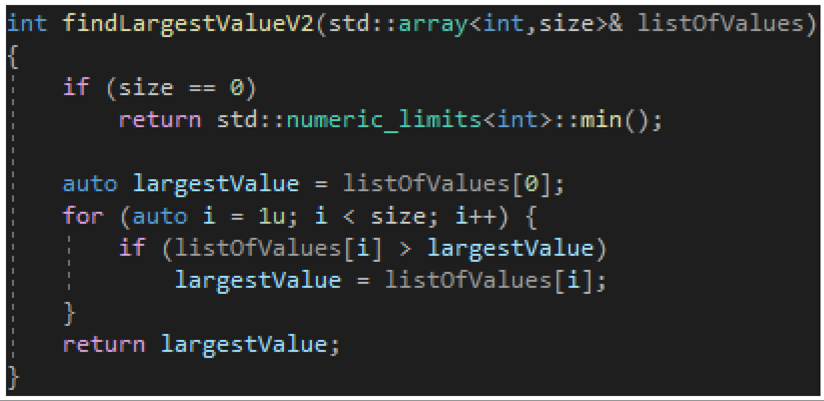
Q5. Span and Arrays

Question:

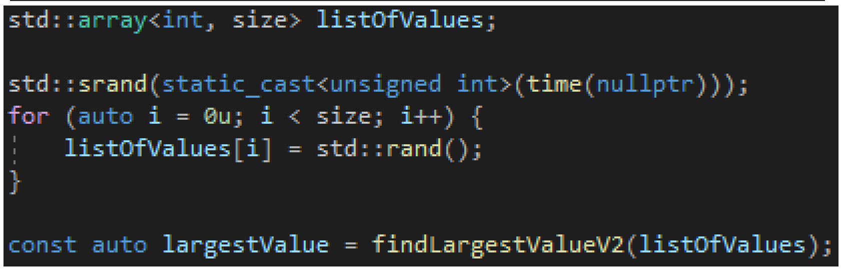
1. Create a version of the code that uses the array template, which wraps a vanilla C array within a C++11 template.
2. Create a version of this function that uses the C++20 span to pass the array to the function. span removes the need for the second parameter

Solution:

1.

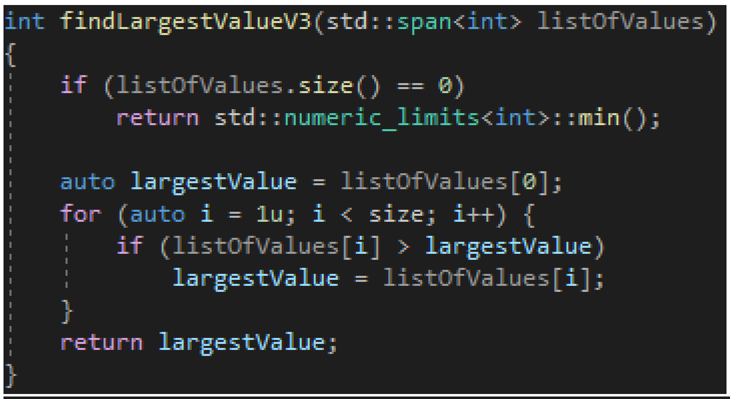


```
int findLargestValueV2(std::array<int,size>& listOfValues)
{
    if (size == 0)
        return std::numeric_limits<int>::min();

    auto largestValue = listOfValues[0];
    for (auto i = 1u; i < size; i++) {
        if (listOfValues[i] > largestValue)
            largestValue = listOfValues[i];
    }
    return largestValue;
}
```

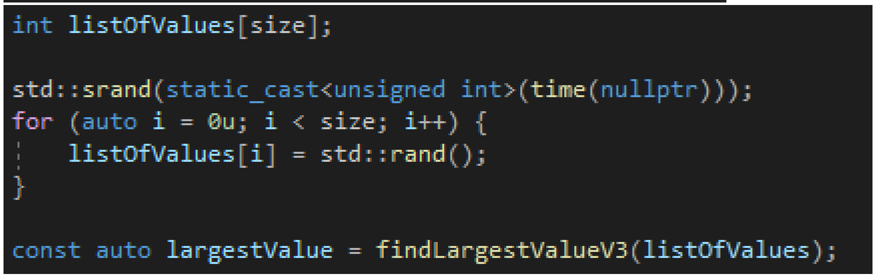


```
std::array<int, size> listOfValues;

std::srand(static_cast<unsigned int>(time(nullptr)));
for (auto i = 0u; i < size; i++) {
    listOfValues[i] = std::rand();
}

const auto largestValue = findLargestValueV2(listOfValues);
```

2.



```
int findLargestValueV3(std::span<int> listOfValues)
{
    if (listOfValues.size() == 0)
        return std::numeric_limits<int>::min();

    auto largestValue = listOfValues[0];
    for (auto i = 1u; i < size; i++) {
        if (listOfValues[i] > largestValue)
            largestValue = listOfValues[i];
    }
    return largestValue;
}
```



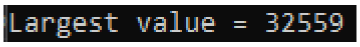
```
int listOfValues[size];

std::srand(static_cast<unsigned int>(time(nullptr)));
for (auto i = 0u; i < size; i++) {
    listOfValues[i] = std::rand();
}

const auto largestValue = findLargestValueV3(listOfValues);
```

Test data:

n/a

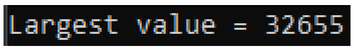
Sample output:

1.



```
Largest value = 32559
```

2.



```
Largest value = 32655
```

Reflection:

Fairly straight forward there isn’t much to comment on. Span defiantly seems like it would be the way I would want to do it if it’s possible to.

Metadata:

Arrays

Further information:

n/a

Week 5 – Lab E
Date: 01/11/2022

Q1. Basic vectors

Question:

1. Add cross product and dot product methods to the given ‘Vector3d’ class.

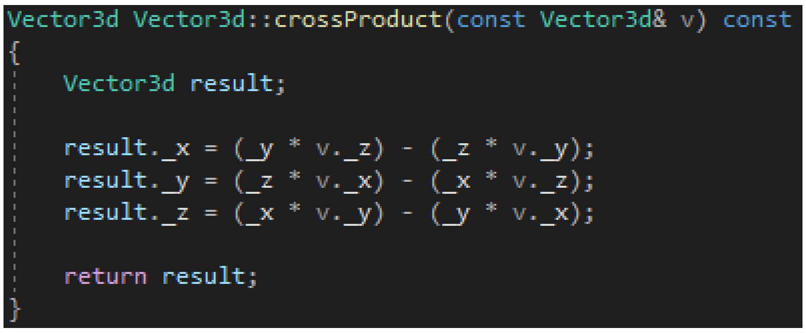
2. Write new methods to overload the basic binary operators.

3. Overload the ‘-‘ unary operator so that it inverts the vector.

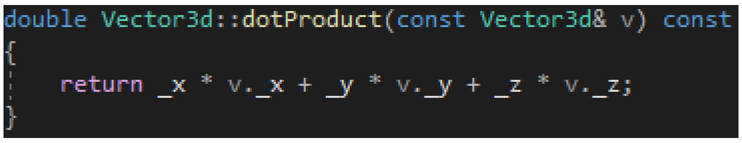
4. Use timing code to analyse the performance of each implementation of adding 2 vectors.

Solution:

1.

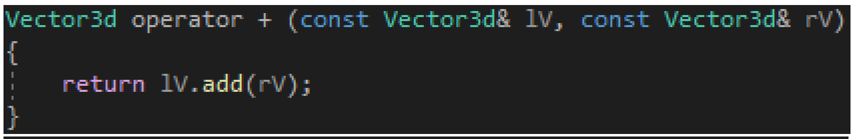


```
Vector3d Vector3d::crossProduct(const Vector3d& v) const
{
    Vector3d result;

    result._x = (_y * v._z) - (_z * v._y);
    result._y = (_z * v._x) - (_x * v._z);
    result._z = (_x * v._y) - (_y * v._x);

    return result;
}
```

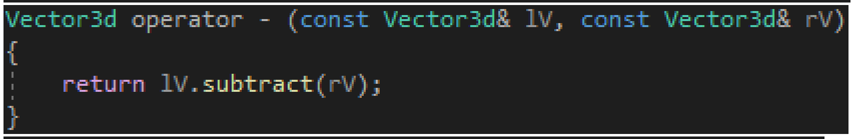


```
double Vector3d::dotProduct(const Vector3d& v) const
{
    return _x * v._x + _y * v._y + _z * v._z;
}
```

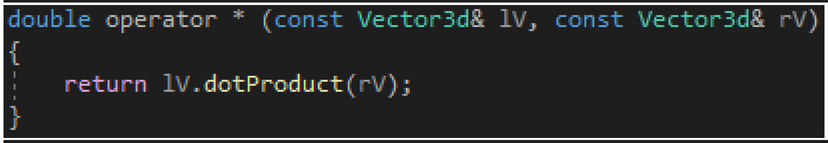
2.



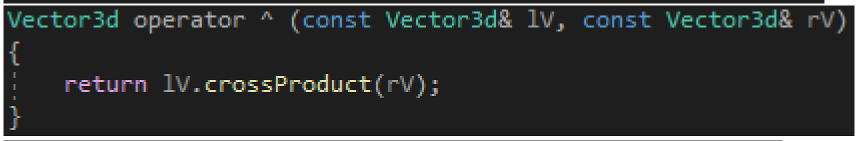
```
Vector3d operator + (const Vector3d& lV, const Vector3d& rV)
{
    return lV.add(rV);
}
```



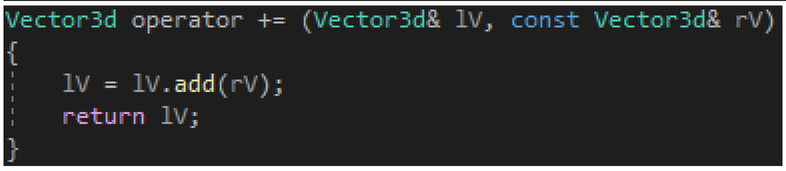
```
Vector3d operator - (const Vector3d& lV, const Vector3d& rV)
{
    return lV.subtract(rV);
}
```



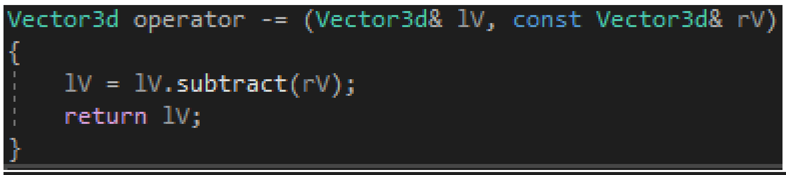
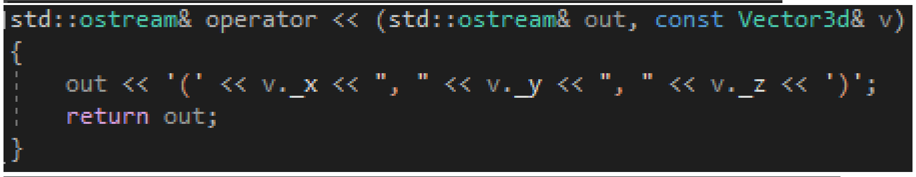
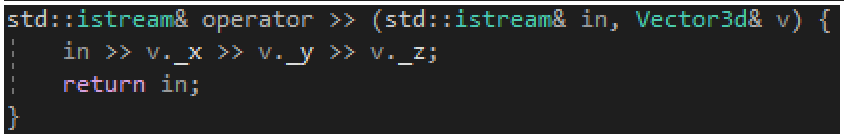
```
double operator * (const Vector3d& lV, const Vector3d& rV)
{
    return lV.dotProduct(rV);
}
```

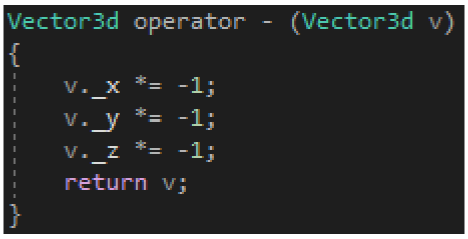


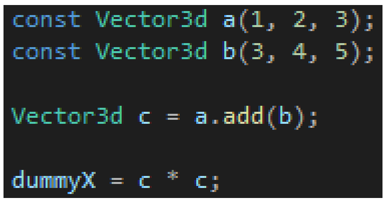
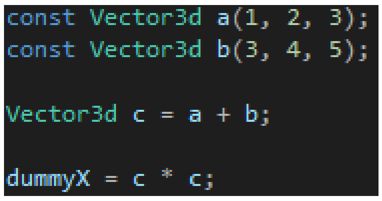
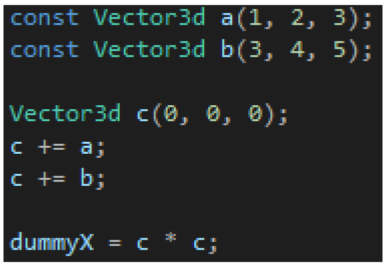
```
Vector3d operator ^ (const Vector3d& lV, const Vector3d& rV)
{
    return lV.crossProduct(rV);
}
```

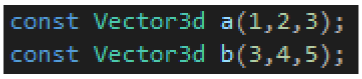


```
Vector3d operator += (Vector3d& lV, const Vector3d& rV)
{
    lV = lV.add(rV);
    return lV;
}
```

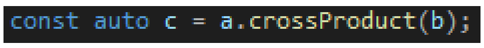

```
Vector3d operator -= (Vector3d& lV, const Vector3d& rV)
{
    lV = lV.subtract(rV);
    return lV;
}
```

```
std::ostream& operator << (std::ostream& out, const Vector3d& v)
{
    out << '(' << v._x << ", " << v._y << ", " << v._z << ')';
    return out;
}
```

```
std::istream& operator >> (std::istream& in, Vector3d& v) {
    in >> v._x >> v._y >> v._z;
    return in;
}
```
3.

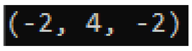
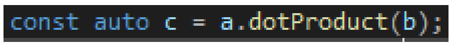

```
Vector3d operator - (Vector3d v)
{
    v._x *= -1;
    v._y *= -1;
    v._z *= -1;
    return v;
}
```
4.

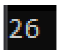

```
const Vector3d a(1, 2, 3);
const Vector3d b(3, 4, 5);

Vector3d c = a.add(b);

dummyX = c * c;
```

```
const Vector3d a(1, 2, 3);
const Vector3d b(3, 4, 5);

Vector3d c = a + b;

dummyX = c * c;
```

```
const Vector3d a(1, 2, 3);
const Vector3d b(3, 4, 5);

Vector3d c(0, 0, 0);
c += a;
c += b;

dummyX = c * c;
```
Test data:

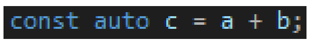

```
const Vector3d a(1,2,3);
const Vector3d b(3,4,5);
```
Sample output:

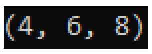
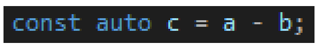
1.

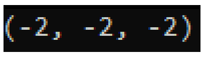
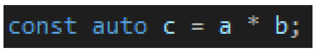

```
const auto c = a.crossProduct(b);
```
=

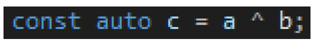

```
(-2, 4, -2)
```

```
const auto c = a.dotProduct(b);
```
=

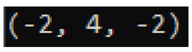

```
26
```
2.

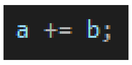

```
const auto c = a + b;
```
=


```
(4, 6, 8)
```

```
const auto c = a - b;
```
=


```
(-2, -2, -2)
```

```
const auto c = a * b;
```
=

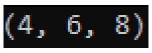
```
26
```

```
const auto c = a ^ b;
```
=


```
(-2, 4, -2)
```

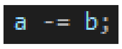


```
a += b;
```

=

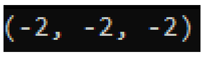


```
(4, 6, 8)
```

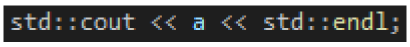


```
a -= b;
```

=

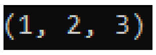


```
(-2, -2, -2)
```

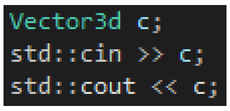


```
std::cout << a << std::endl;
```

=

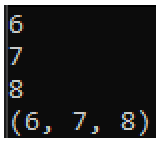


```
(1, 2, 3)
```



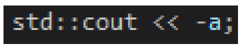
```
Vector3d c;
std::cin >> c;
std::cout << c;
```

=



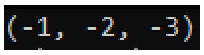
```
6
7
8
(6, 7, 8)
```

3.



```
std::cout << -a;
```

=

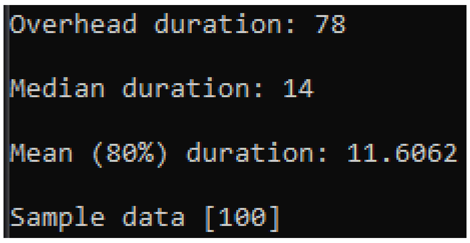


```
(-1, -2, -3)
```

4.

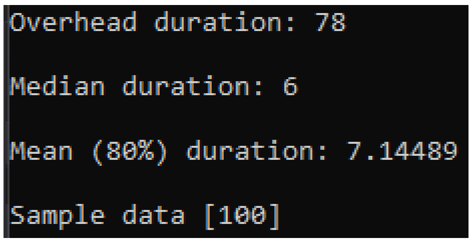
Release x64

.add()



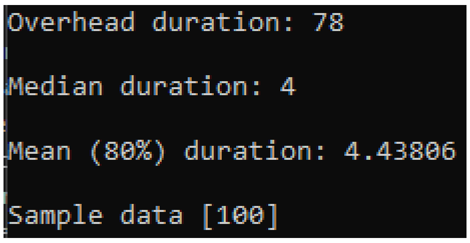
```
Overhead duration: 78

Median duration: 14

Mean (80%) duration: 11.6062

Sample data [100]
```

+



```
Overhead duration: 78

Median duration: 6

Mean (80%) duration: 7.14489

Sample data [100]
```

+=



```
Overhead duration: 78

Median duration: 4

Mean (80%) duration: 4.43806

Sample data [100]
```

Reflection:

This was a pretty easy task to do. In terms of the timings for the addition it is as I would have expected for the most part with operators being faster weirdly enough though += seems to be faster than +.

Metadata:

Vector maths

Further information:

Some methods are inline to remove the associated overhead which will improve execution time

Why exactly are operators faster than a method? The operators are calling the method anyway

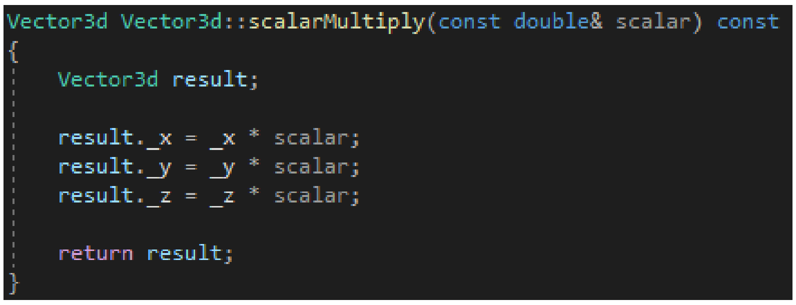
Why is += faster the +?

Q2. Commutativity

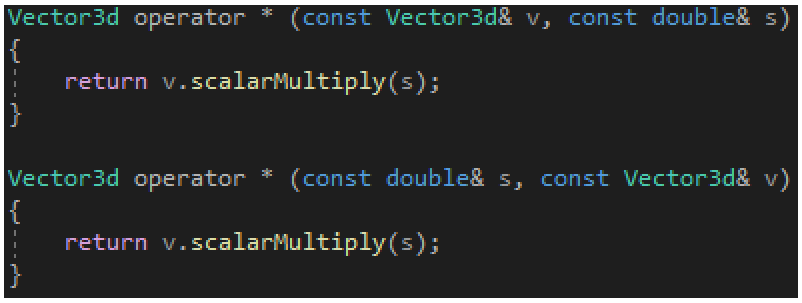
Question:

Implement both a standard method and overload the * operator to multiply a vector by a single double. Also implement the multiplication of a single double by a vector.

Solution:

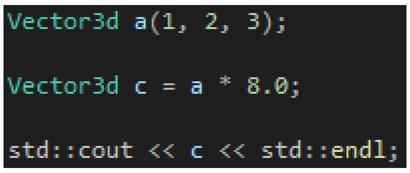


```
Vector3d Vector3d::scalarMultiply(const double& scalar) const
{
    Vector3d result;

    result._x = _x * scalar;
    result._y = _y * scalar;
    result._z = _z * scalar;

    return result;
}
```

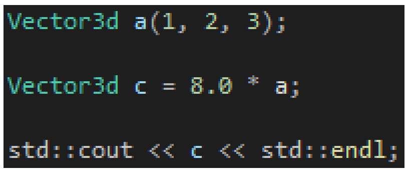


```
Vector3d operator * (const Vector3d& v, const double& s)
{
    return v.scalarMultiply(s);
}

Vector3d operator * (const double& s, const Vector3d& v)
{
    return v.scalarMultiply(s);
}
```

Test data:

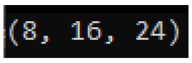


```
Vector3d a(1, 2, 3);

Vector3d c = a * 8.0;

std::cout << c << std::endl;
```



```
Vector3d a(1, 2, 3);

Vector3d c = 8.0 * a;

std::cout << c << std::endl;
```

Sample output:



```
(8, 16, 24)
```

Reflection:

This was pretty straight forward there isn't a lot to really comment on

Metadata:

Scalar multiply

Further information:

n/a

Q3. Matrices

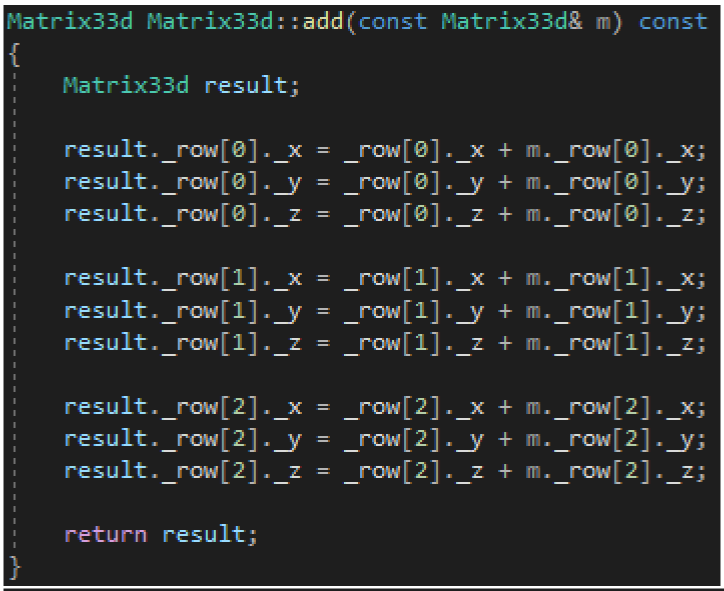
Question:

Add the functionality to do the following:

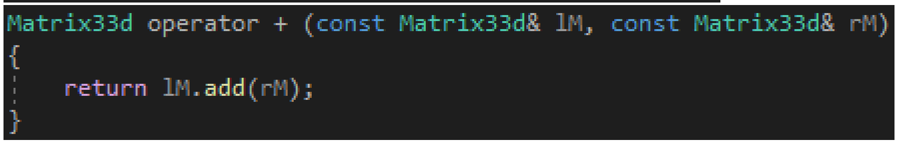
1. Addition
2. Subtraction
3. Multiplication
4. Streaming in and out
5. Inverse
6. Transpose

Solution:

1.

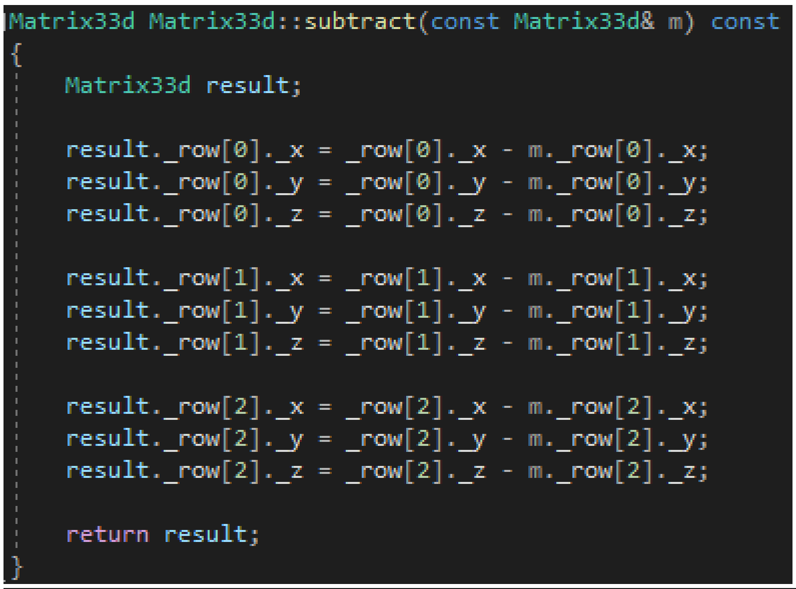


```
Matrix33d Matrix33d::add(const Matrix33d& m) const
{
    Matrix33d result;

    result._row[0]._x = _row[0]._x + m._row[0]._x;
    result._row[0]._y = _row[0]._y + m._row[0]._y;
    result._row[0]._z = _row[0]._z + m._row[0]._z;

    result._row[1]._x = _row[1]._x + m._row[1]._x;
    result._row[1]._y = _row[1]._y + m._row[1]._y;
    result._row[1]._z = _row[1]._z + m._row[1]._z;

    result._row[2]._x = _row[2]._x + m._row[2]._x;
    result._row[2]._y = _row[2]._y + m._row[2]._y;
    result._row[2]._z = _row[2]._z + m._row[2]._z;

    return result;
}
```

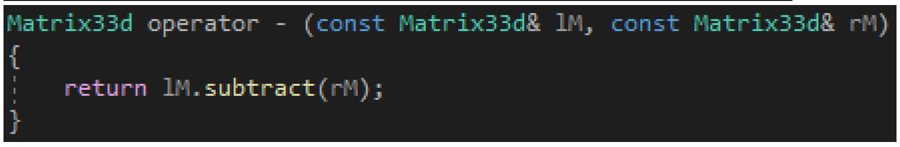


```
Matrix33d operator + (const Matrix33d& lM, const Matrix33d& rM)
{
    return lM.add(rM);
}
```

2.

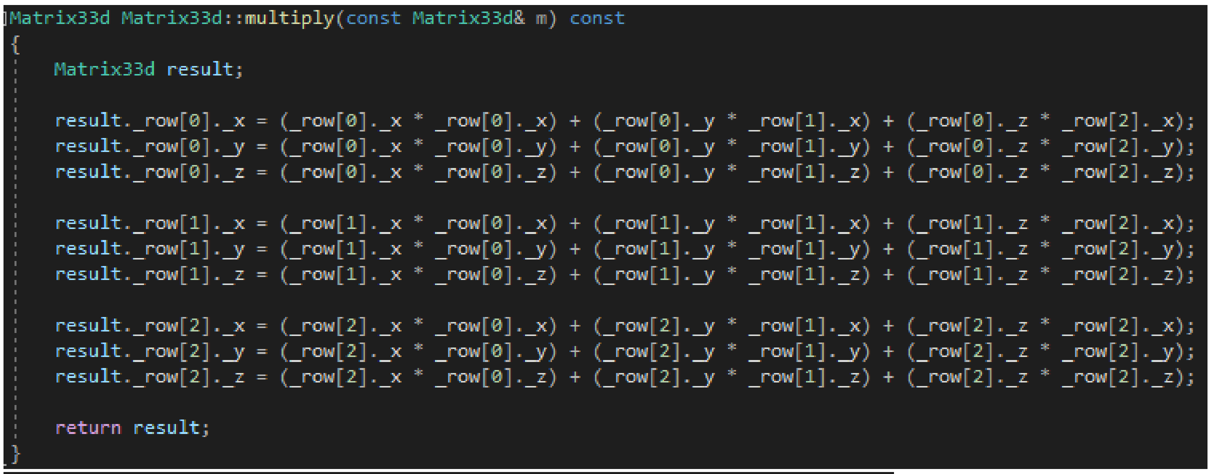


```cpp
Matrix33d Matrix33d::subtract(const Matrix33d& m) const
{
    Matrix33d result;

    result._row[0]._x = _row[0]._x - m._row[0]._x;
    result._row[0]._y = _row[0]._y - m._row[0]._y;
    result._row[0]._z = _row[0]._z - m._row[0]._z;

    result._row[1]._x = _row[1]._x - m._row[1]._x;
    result._row[1]._y = _row[1]._y - m._row[1]._y;
    result._row[1]._z = _row[1]._z - m._row[1]._z;

    result._row[2]._x = _row[2]._x - m._row[2]._x;
    result._row[2]._y = _row[2]._y - m._row[2]._y;
    result._row[2]._z = _row[2]._z - m._row[2]._z;

    return result;
}
```

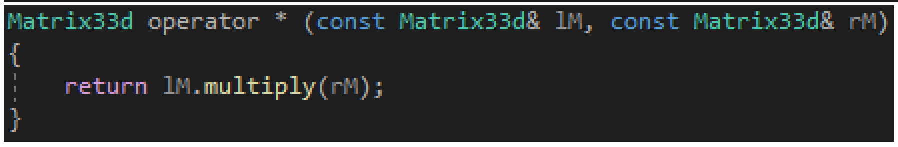


```cpp
Matrix33d operator - (const Matrix33d& lM, const Matrix33d& rM)
{
    return lM.subtract(rM);
}
```

3.

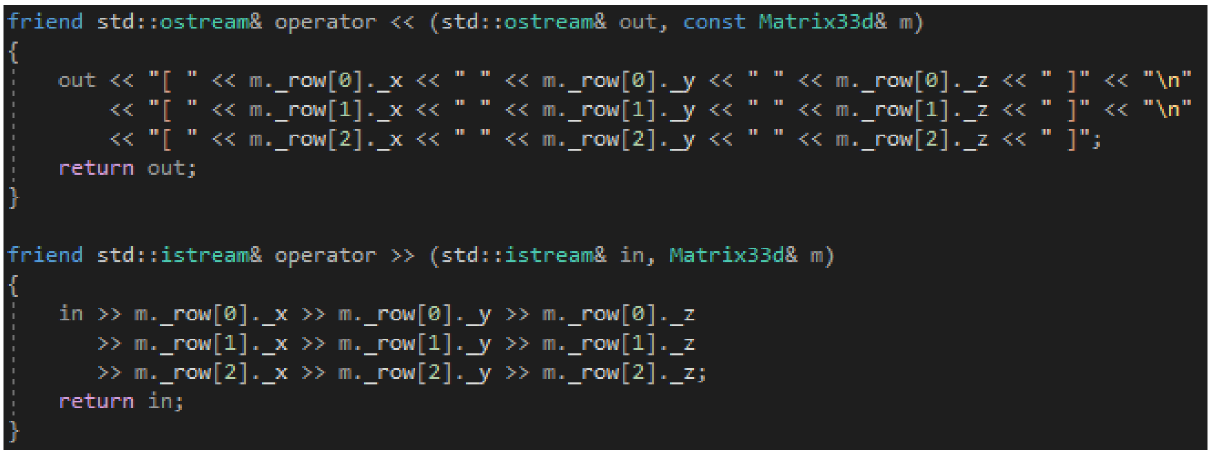


```cpp
Matrix33d Matrix33d::multiply(const Matrix33d& m) const
{
    Matrix33d result;

    result._row[0]._x = (_row[0]._x * _row[0]._x) + (_row[0]._y * _row[1]._x) + (_row[0]._z * _row[2]._x);
    result._row[0]._y = (_row[0]._x * _row[0]._y) + (_row[0]._y * _row[1]._y) + (_row[0]._z * _row[2]._y);
    result._row[0]._z = (_row[0]._x * _row[0]._z) + (_row[0]._y * _row[1]._z) + (_row[0]._z * _row[2]._z);

    result._row[1]._x = (_row[1]._x * _row[0]._x) + (_row[1]._y * _row[1]._x) + (_row[1]._z * _row[2]._x);
    result._row[1]._y = (_row[1]._x * _row[0]._y) + (_row[1]._y * _row[1]._y) + (_row[1]._z * _row[2]._y);
    result._row[1]._z = (_row[1]._x * _row[0]._z) + (_row[1]._y * _row[1]._z) + (_row[1]._z * _row[2]._z);

    result._row[2]._x = (_row[2]._x * _row[0]._x) + (_row[2]._y * _row[1]._x) + (_row[2]._z * _row[2]._x);
    result._row[2]._y = (_row[2]._x * _row[0]._y) + (_row[2]._y * _row[1]._y) + (_row[2]._z * _row[2]._y);
    result._row[2]._z = (_row[2]._x * _row[0]._z) + (_row[2]._y * _row[1]._z) + (_row[2]._z * _row[2]._z);

    return result;
}
```



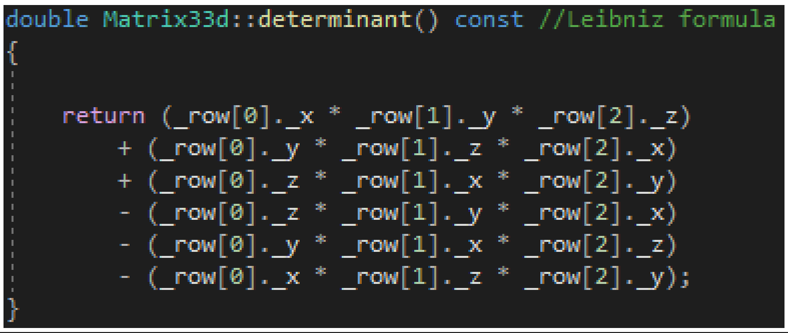
```cpp
Matrix33d operator * (const Matrix33d& lM, const Matrix33d& rM)
{
    return lM.multiply(rM);
}
```

4.

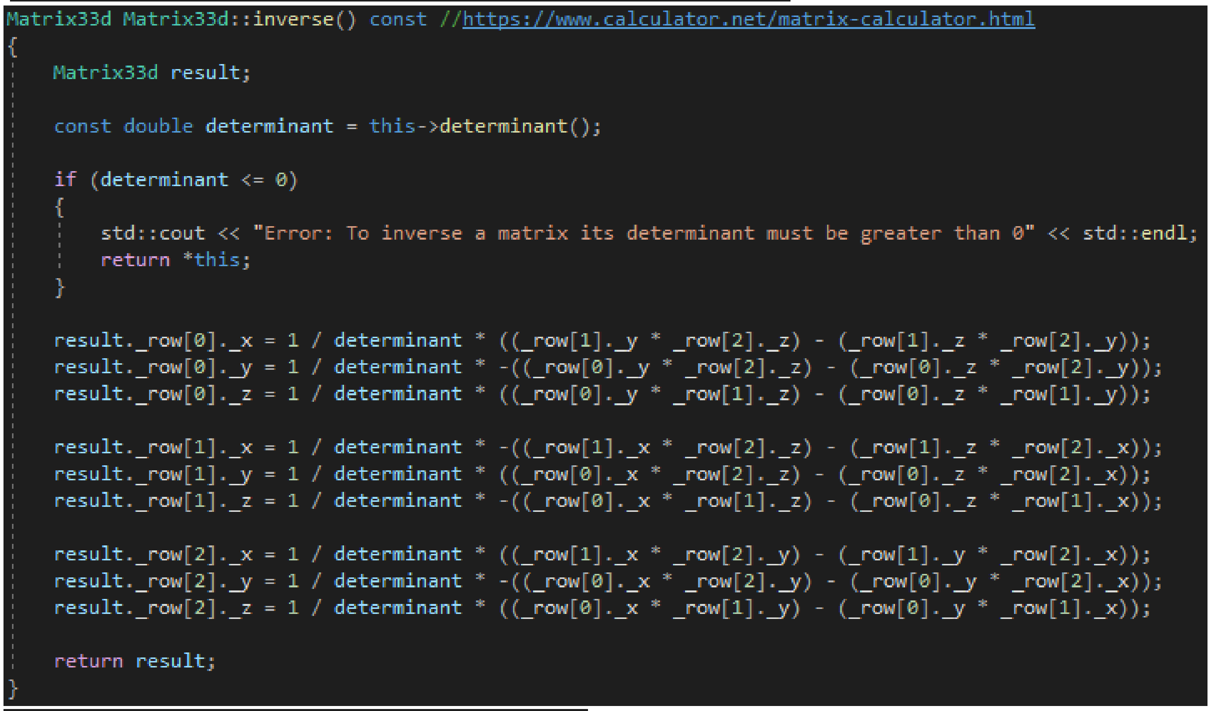


```cpp
friend std::ostream& operator << (std::ostream& out, const Matrix33d& m)
{
    out << "[ " << m._row[0]._x << " " << m._row[0]._y << " " << m._row[0]._z << " ]" << "\n"
        << "[ " << m._row[1]._x << " " << m._row[1]._y << " " << m._row[1]._z << " ]" << "\n"
        << "[ " << m._row[2]._x << " " << m._row[2]._y << " " << m._row[2]._z << " ]";
    return out;
}

friend std::istream& operator >> (std::istream& in, Matrix33d& m)
{
    in >> m._row[0]._x >> m._row[0]._y >> m._row[0]._z
       >> m._row[1]._x >> m._row[1]._y >> m._row[1]._z
       >> m._row[2]._x >> m._row[2]._y >> m._row[2]._z;
    return in;
}
```

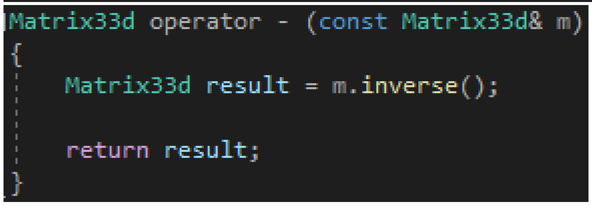
5.



```cpp
double Matrix33d::determinant() const //Leibniz formula
{

    return (_row[0]._x * _row[1]._y * _row[2]._z)
        + (_row[0]._y * _row[1]._z * _row[2]._x)
        + (_row[0]._z * _row[1]._x * _row[2]._y)
        - (_row[0]._z * _row[1]._y * _row[2]._x)
        - (_row[0]._y * _row[1]._x * _row[2]._z)
        - (_row[0]._x * _row[1]._z * _row[2]._y);
}
```

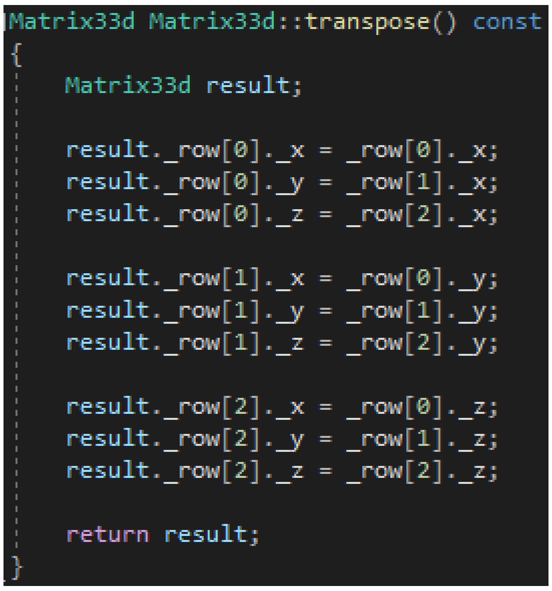


```cpp
Matrix33d Matrix33d::inverse() const //https://www.calculator.net/matrix-calculator.html
{
    Matrix33d result;

    const double determinant = this->determinant();

    if (determinant <= 0)
    {
        std::cout << "Error: To inverse a matrix its determinant must be greater than 0" << std::endl;
        return *this;
    }

    result._row[0]._x = 1 / determinant * ((_row[1]._y * _row[2]._z) - (_row[1]._z * _row[2]._y));
    result._row[0]._y = 1 / determinant * -((_row[0]._y * _row[2]._z) - (_row[0]._z * _row[2]._y));
    result._row[0]._z = 1 / determinant * ((_row[0]._y * _row[1]._z) - (_row[0]._z * _row[1]._y));

    result._row[1]._x = 1 / determinant * -((_row[1]._x * _row[2]._z) - (_row[1]._z * _row[2]._x));
    result._row[1]._y = 1 / determinant * ((_row[0]._x * _row[2]._z) - (_row[0]._z * _row[2]._x));
    result._row[1]._z = 1 / determinant * -((_row[0]._x * _row[1]._z) - (_row[0]._z * _row[1]._x));

    result._row[2]._x = 1 / determinant * ((_row[1]._x * _row[2]._y) - (_row[1]._y * _row[2]._x));
    result._row[2]._y = 1 / determinant * -((_row[0]._x * _row[2]._y) - (_row[0]._y * _row[2]._x));
    result._row[2]._z = 1 / determinant * ((_row[0]._x * _row[1]._y) - (_row[0]._y * _row[1]._x));

    return result;
}
```



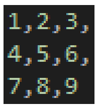
```cpp
Matrix33d operator - (const Matrix33d& m)
{
    Matrix33d result = m.inverse();

    return result;
}
```

6.

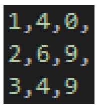


```
Matrix33d Matrix33d::transpose() const
{
    Matrix33d result;

    result._row[0]._x = _row[0]._x;
    result._row[0]._y = _row[1]._x;
    result._row[0]._z = _row[2]._x;

    result._row[1]._x = _row[0]._y;
    result._row[1]._y = _row[1]._y;
    result._row[1]._z = _row[2]._y;

    result._row[2]._x = _row[0]._z;
    result._row[2]._y = _row[1]._z;
    result._row[2]._z = _row[2]._z;

    return result;
}
```

Test data:

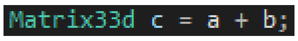


```
1,2,3,
4,5,6,
7,8,9
```



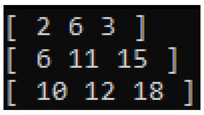
```
1,4,0,
2,6,9,
3,4,9
```

Sample output:

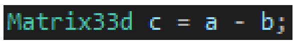


```
Matrix33d c = a + b;
```

=

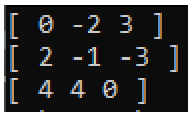


```
[ 2 6 3 ]
[ 6 11 15 ]
[ 10 12 18 ]
```

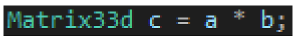


```
Matrix33d c = a - b;
```

=

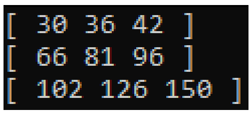


```
[ 0 -2 3 ]
[ 2 -1 -3 ]
[ 4 4 0 ]
```

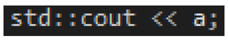


```
Matrix33d c = a * b;
```

=

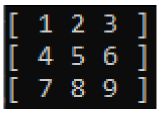


```
[ 30 36 42 ]
[ 66 81 96 ]
[ 102 126 150 ]
```

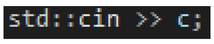


```
std::cout << a;
```

=

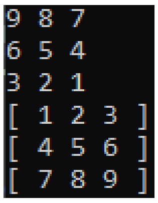


```
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
```

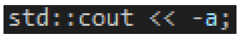


```
std::cin >> c;
```

=

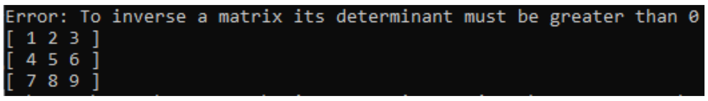


```
9 8 7
6 5 4
3 2 1
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
```

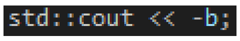


```
std::cout << -a;
```

=

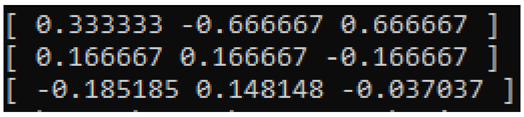


```
Error: To inverse a matrix its determinant must be greater than 0
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
```

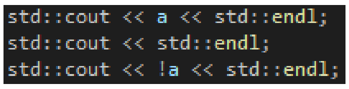


```
std::cout << -b;
```

=

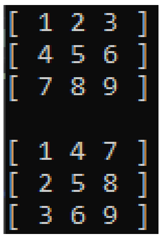


```
[ 0.333333 -0.666667 0.666667 ]
[ 0.166667 0.166667 -0.166667 ]
[ -0.185185 0.148148 -0.037037 ]
```



```
std::cout << a << std::endl;
std::cout << std::endl;
std::cout << !a << std::endl;
```

=



```
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]

[ 1 4 7 ]
[ 2 5 8 ]
[ 3 6 9 ]
```

Reflection:

This was a complicated task purely because of some of the maths involved especially inversing. I struggled to find an appropriate operator to overload for transposing. Is it sometimes better to use a method call over an operator? Operators are quicker but in the case of the transpose it can be less clear what is going on.

Metadata:

Matrix maths

Further information:

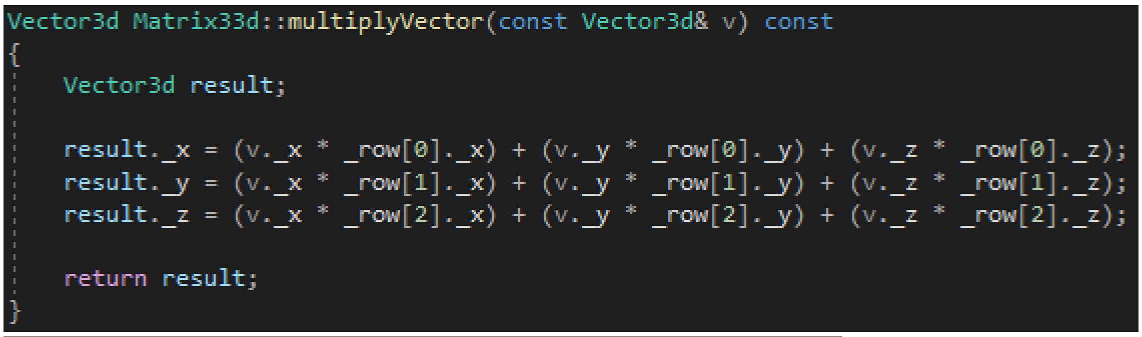
Why does `double determinant = this->determinant();` work but `double determinant = this.determinant();` doesn't? I thought . and -> were the same, is it because is a method call within a class

Q4. Vector and Matrix Multiplication

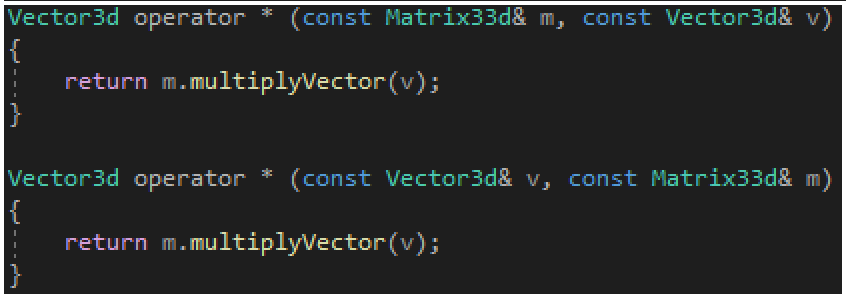
Question:

Expand your Matrix33d class to be able to multiple a Vector3d object by a Matrix33d object

Solution:

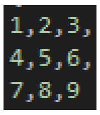


```
Vector3d Matrix33d::multiplyVector(const Vector3d& v) const
{
    Vector3d result;

    result._x = (v._x * _row[0]._x) + (v._y * _row[0]._y) + (v._z * _row[0]._z);
    result._y = (v._x * _row[1]._x) + (v._y * _row[1]._y) + (v._z * _row[1]._z);
    result._z = (v._x * _row[2]._x) + (v._y * _row[2]._y) + (v._z * _row[2]._z);

    return result;
}
```

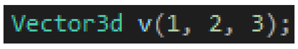


```
Vector3d operator * (const Matrix33d& m, const Vector3d& v)
{
    return m.multiplyVector(v);
}

Vector3d operator * (const Vector3d& v, const Matrix33d& m)
{
    return m.multiplyVector(v);
}
```

Test data:

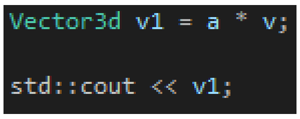


```
1,2,3,
4,5,6,
7,8,9
```



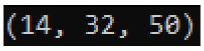
```
Vector3d v(1, 2, 3);
```

Sample output:

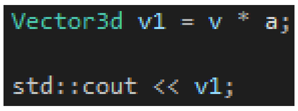


```
Vector3d v1 = a * v;

std::cout << v1;
```

=

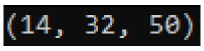


```
(14, 32, 50)
```



```
Vector3d v1 = v * a;

std::cout << v1;
```

=



```
(14, 32, 50)
```

Reflection:

This was very simple assuming I've understood the task correctly. It seems too simple so I have a feeling I've misunderstood something

Metadata:

Vector * Matrix

Further information:

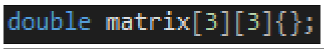
n/a

Q5. Internal data structures

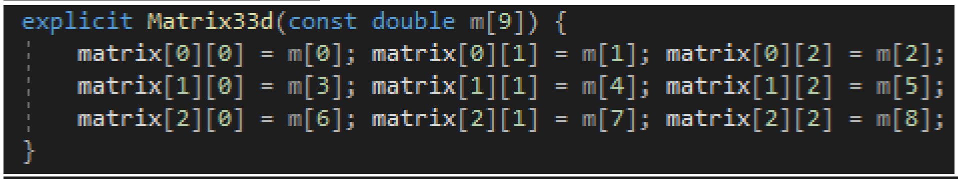
Question:

implement the Matrix33d using a different data format and assess the performance using the timing code from earlier labs

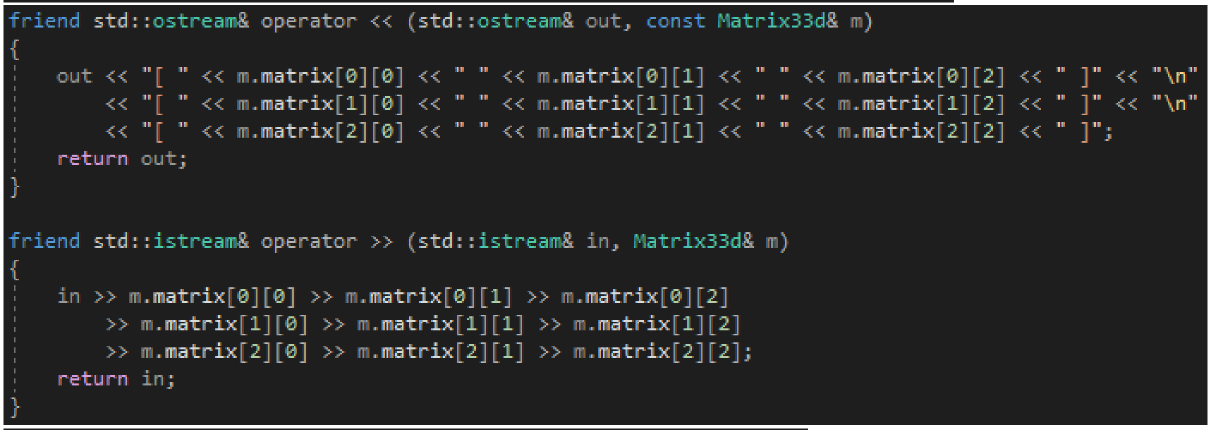
Solution:



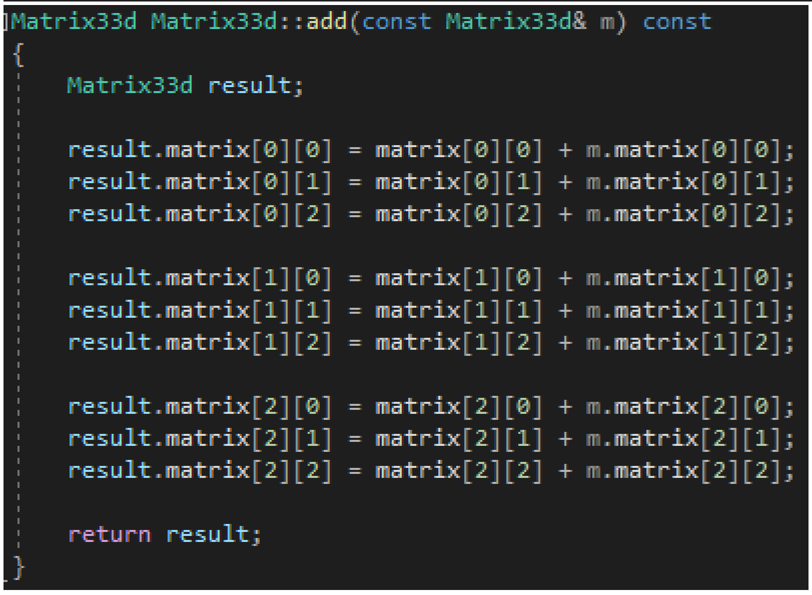
```cpp
double matrix[3][3]{};
```



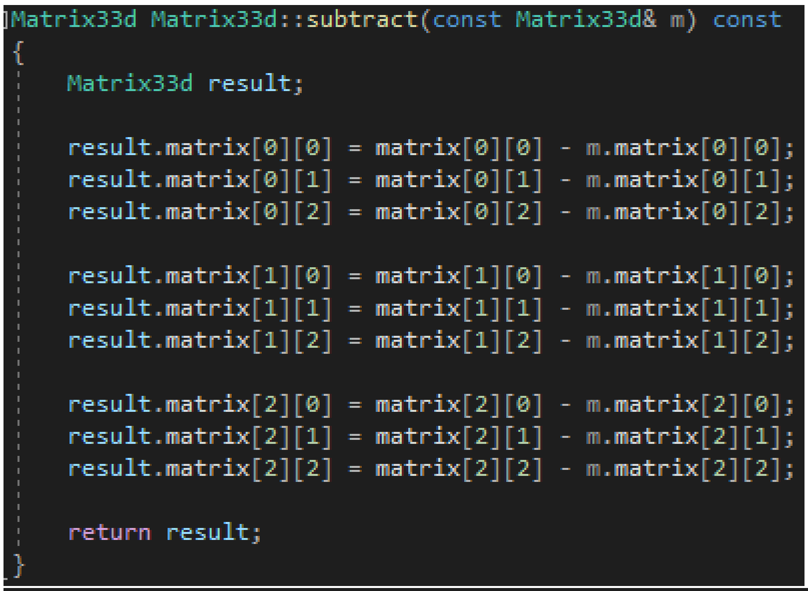
```cpp
explicit Matrix33d(const double m[9]) {
    matrix[0][0] = m[0]; matrix[0][1] = m[1]; matrix[0][2] = m[2];
    matrix[1][0] = m[3]; matrix[1][1] = m[4]; matrix[1][2] = m[5];
    matrix[2][0] = m[6]; matrix[2][1] = m[7]; matrix[2][2] = m[8];
}
```



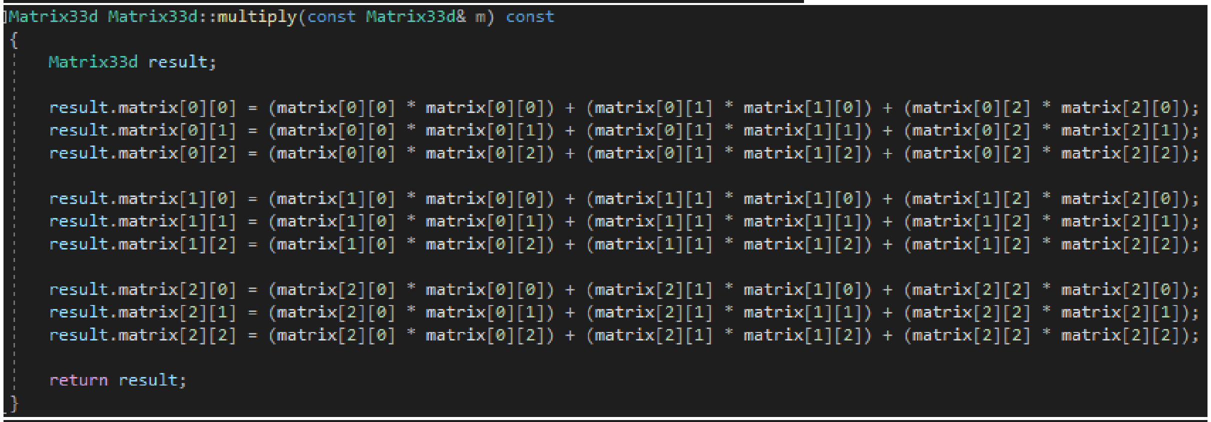
```cpp
friend std::ostream& operator << (std::ostream& out, const Matrix33d& m)
{
    out << "[ " << m.matrix[0][0] << " " << m.matrix[0][1] << " " << m.matrix[0][2] << " ]" << "\n"
        << "[ " << m.matrix[1][0] << " " << m.matrix[1][1] << " " << m.matrix[1][2] << " ]" << "\n"
        << "[ " << m.matrix[2][0] << " " << m.matrix[2][1] << " " << m.matrix[2][2] << " ]";
    return out;
}

friend std::istream& operator >> (std::istream& in, Matrix33d& m)
{
    in >> m.matrix[0][0] >> m.matrix[0][1] >> m.matrix[0][2]
        >> m.matrix[1][0] >> m.matrix[1][1] >> m.matrix[1][2]
        >> m.matrix[2][0] >> m.matrix[2][1] >> m.matrix[2][2];
    return in;
}
```



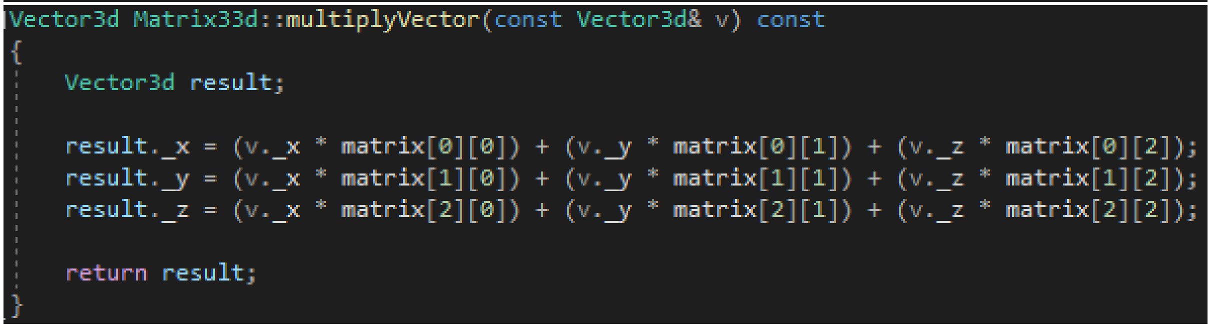
```cpp
Matrix33d Matrix33d::add(const Matrix33d& m) const
{
    Matrix33d result;

    result.matrix[0][0] = matrix[0][0] + m.matrix[0][0];
    result.matrix[0][1] = matrix[0][1] + m.matrix[0][1];
    result.matrix[0][2] = matrix[0][2] + m.matrix[0][2];

    result.matrix[1][0] = matrix[1][0] + m.matrix[1][0];
    result.matrix[1][1] = matrix[1][1] + m.matrix[1][1];
    result.matrix[1][2] = matrix[1][2] + m.matrix[1][2];

    result.matrix[2][0] = matrix[2][0] + m.matrix[2][0];
    result.matrix[2][1] = matrix[2][1] + m.matrix[2][1];
    result.matrix[2][2] = matrix[2][2] + m.matrix[2][2];

    return result;
}
```



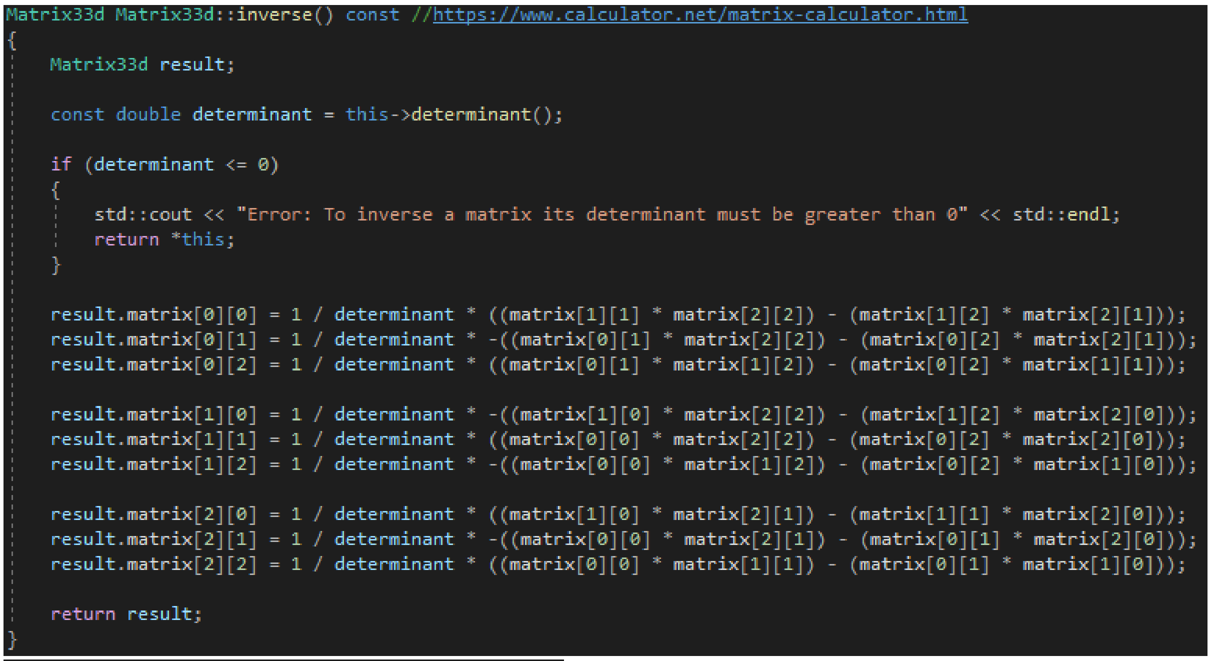
```cpp
Matrix33d Matrix33d::subtract(const Matrix33d& m) const
{
    Matrix33d result;

    result.matrix[0][0] = matrix[0][0] - m.matrix[0][0];
    result.matrix[0][1] = matrix[0][1] - m.matrix[0][1];
    result.matrix[0][2] = matrix[0][2] - m.matrix[0][2];

    result.matrix[1][0] = matrix[1][0] - m.matrix[1][0];
    result.matrix[1][1] = matrix[1][1] - m.matrix[1][1];
    result.matrix[1][2] = matrix[1][2] - m.matrix[1][2];

    result.matrix[2][0] = matrix[2][0] - m.matrix[2][0];
    result.matrix[2][1] = matrix[2][1] - m.matrix[2][1];
    result.matrix[2][2] = matrix[2][2] - m.matrix[2][2];

    return result;
}
```



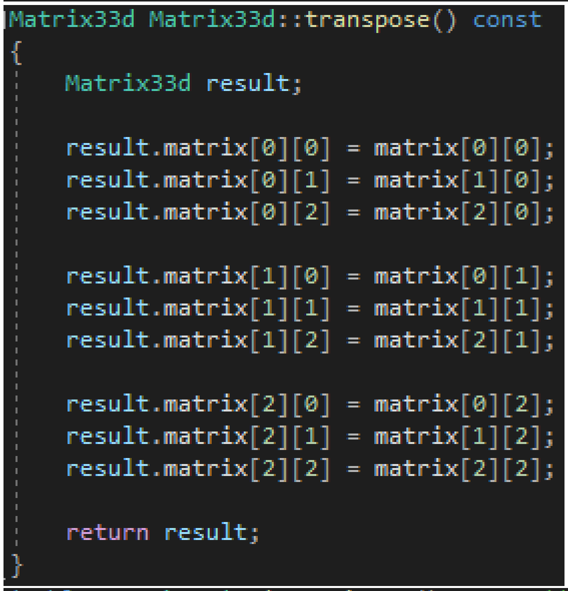
```cpp
Matrix33d Matrix33d::multiply(const Matrix33d& m) const
{
    Matrix33d result;

    result.matrix[0][0] = (matrix[0][0] * matrix[0][0]) + (matrix[0][1] * matrix[1][0]) + (matrix[0][2] * matrix[2][0]);
    result.matrix[0][1] = (matrix[0][0] * matrix[0][1]) + (matrix[0][1] * matrix[1][1]) + (matrix[0][2] * matrix[2][1]);
    result.matrix[0][2] = (matrix[0][0] * matrix[0][2]) + (matrix[0][1] * matrix[1][2]) + (matrix[0][2] * matrix[2][2]);

    result.matrix[1][0] = (matrix[1][0] * matrix[0][0]) + (matrix[1][1] * matrix[1][0]) + (matrix[1][2] * matrix[2][0]);
    result.matrix[1][1] = (matrix[1][0] * matrix[0][1]) + (matrix[1][1] * matrix[1][1]) + (matrix[1][2] * matrix[2][1]);
    result.matrix[1][2] = (matrix[1][0] * matrix[0][2]) + (matrix[1][1] * matrix[1][2]) + (matrix[1][2] * matrix[2][2]);

    result.matrix[2][0] = (matrix[2][0] * matrix[0][0]) + (matrix[2][1] * matrix[1][0]) + (matrix[2][2] * matrix[2][0]);
    result.matrix[2][1] = (matrix[2][0] * matrix[0][1]) + (matrix[2][1] * matrix[1][1]) + (matrix[2][2] * matrix[2][1]);
    result.matrix[2][2] = (matrix[2][0] * matrix[0][2]) + (matrix[2][1] * matrix[1][2]) + (matrix[2][2] * matrix[2][2]);

    return result;
}
```



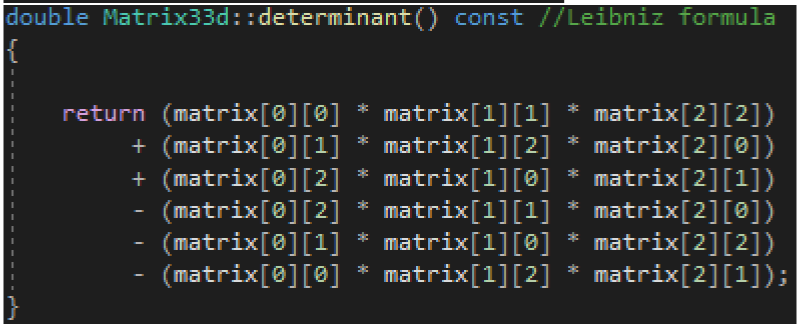
```cpp
Vector3d Matrix33d::multiplyVector(const Vector3d& v) const
{
    Vector3d result;

    result._x = (v._x * matrix[0][0]) + (v._y * matrix[0][1]) + (v._z * matrix[0][2]);
    result._y = (v._x * matrix[1][0]) + (v._y * matrix[1][1]) + (v._z * matrix[1][2]);
    result._z = (v._x * matrix[2][0]) + (v._y * matrix[2][1]) + (v._z * matrix[2][2]);

    return result;
}
```



```
Matrix33d Matrix33d::inverse() const //https://www.calculator.net/matrix-calculator.html
{
    Matrix33d result;

    const double determinant = this->determinant();

    if (determinant <= 0)
    {
        std::cout << "Error: To inverse a matrix its determinant must be greater than 0" << std::endl;
        return *this;
    }

    result.matrix[0][0] = 1 / determinant * ((matrix[1][1] * matrix[2][2]) - (matrix[1][2] * matrix[2][1]));
    result.matrix[0][1] = 1 / determinant * -((matrix[0][1] * matrix[2][2]) - (matrix[0][2] * matrix[2][1]));
    result.matrix[0][2] = 1 / determinant * ((matrix[0][1] * matrix[1][2]) - (matrix[0][2] * matrix[1][1]));

    result.matrix[1][0] = 1 / determinant * -((matrix[1][0] * matrix[2][2]) - (matrix[1][2] * matrix[2][0]));
    result.matrix[1][1] = 1 / determinant * ((matrix[0][0] * matrix[2][2]) - (matrix[0][2] * matrix[2][0]));
    result.matrix[1][2] = 1 / determinant * -((matrix[0][0] * matrix[1][2]) - (matrix[0][2] * matrix[1][0]));

    result.matrix[2][0] = 1 / determinant * ((matrix[1][0] * matrix[2][1]) - (matrix[1][1] * matrix[2][0]));
    result.matrix[2][1] = 1 / determinant * -((matrix[0][0] * matrix[2][1]) - (matrix[0][1] * matrix[2][0]));
    result.matrix[2][2] = 1 / determinant * ((matrix[0][0] * matrix[1][1]) - (matrix[0][1] * matrix[1][0]));

    return result;
}
```



```
Matrix33d Matrix33d::transpose() const
{
    Matrix33d result;

    result.matrix[0][0] = matrix[0][0];
    result.matrix[0][1] = matrix[1][0];
    result.matrix[0][2] = matrix[2][0];

    result.matrix[1][0] = matrix[0][1];
    result.matrix[1][1] = matrix[1][1];
    result.matrix[1][2] = matrix[2][1];

    result.matrix[2][0] = matrix[0][2];
    result.matrix[2][1] = matrix[1][2];
    result.matrix[2][2] = matrix[2][2];

    return result;
}
```



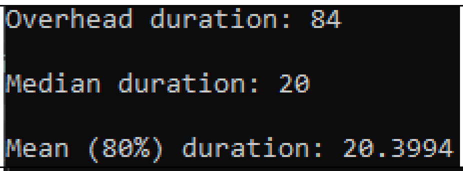
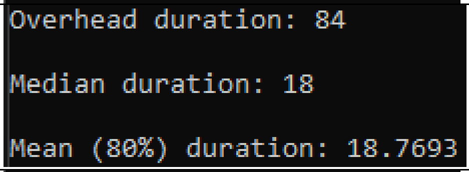
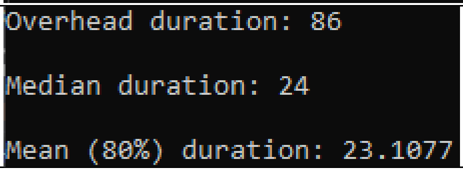
```
double Matrix33d::determinant() const //Leibniz formula
{

    return (matrix[0][0] * matrix[1][1] * matrix[2][2])
         + (matrix[0][1] * matrix[1][2] * matrix[2][0])
         + (matrix[0][2] * matrix[1][0] * matrix[2][1])
         - (matrix[0][2] * matrix[1][1] * matrix[2][0])
         - (matrix[0][1] * matrix[1][0] * matrix[2][2])
         - (matrix[0][0] * matrix[1][2] * matrix[2][1]);
}
```

Test data:

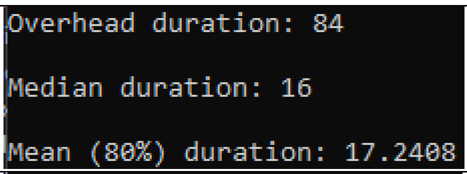
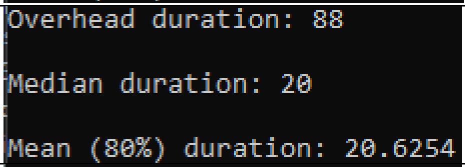
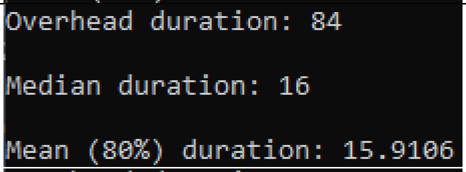
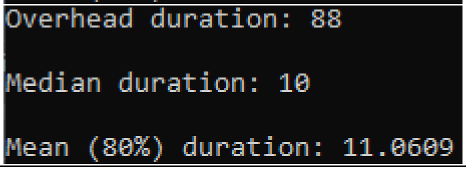
n/a

Sample output:

Array of Vector3d

Addition	 Overhead duration: 84 Median duration: 20 Mean (80%) duration: 20.3994
Subtraction	 Overhead duration: 84 Median duration: 18 Mean (80%) duration: 18.7693
Multiply Matrices	 Overhead duration: 86 Median duration: 24 Mean (80%) duration: 23.1077

Multidimensional array of doubles

Addition	 Overhead duration: 84 Median duration: 16 Mean (80%) duration: 17.2408
Subtraction	 Overhead duration: 88 Median duration: 20 Mean (80%) duration: 20.6254
Multiply Matrices	 Overhead duration: 84 Median duration: 16 Mean (80%) duration: 15.9106  Overhead duration: 88 Median duration: 10 Mean (80%) duration: 11.0609

Reflection:

I decided that the best data structure to implement would be a multidimensional array of doubles, this is essentially the same as the original implementation which was and array of 'Vector3d' object. I think the multidimensional array is a lot easier to read and understand though. When looking at timings it seems multidimensional is quicker but the timing was pretty inconsistent so I'm not sure if it is

Metadata:

Matrix data structures

Further information:

n/a

Week 6 – Lab F

Date: 08/11/2022

Q1. Big strings

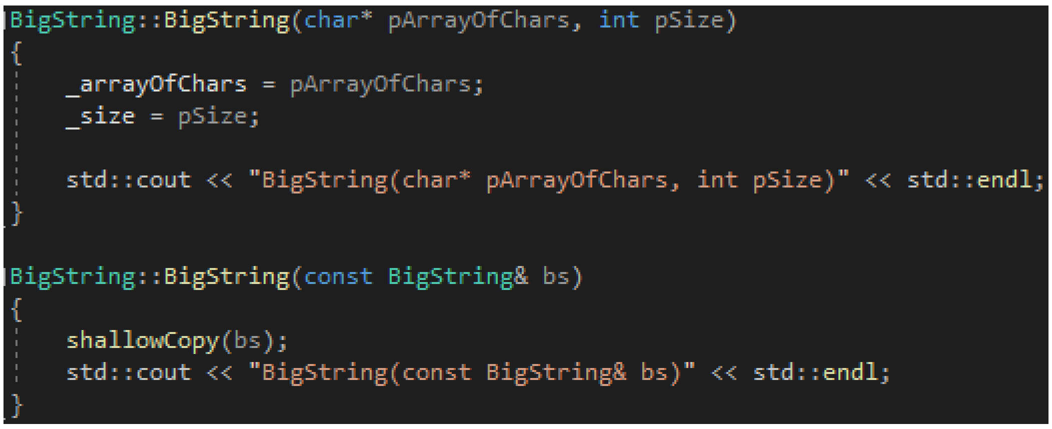
Question:

Expand the “BigString” class to contain at least the following functionality:

1. Constructors
2. Destructor
3. Assignment operator
4. Stream in and out
5. Index operator

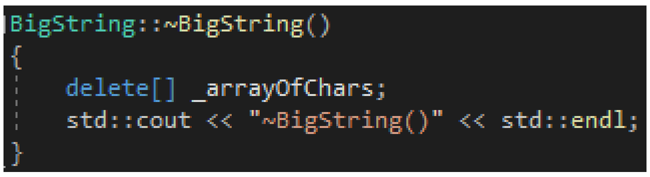
Solution:

1.



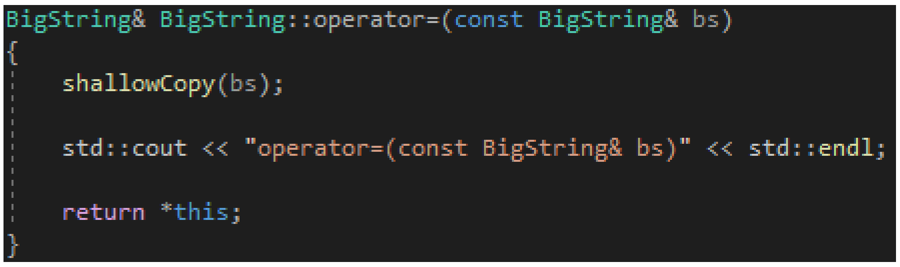
```cpp
BigString::BigString(char* pArrayOfChars, int pSize)
{
    _arrayOfChars = pArrayOfChars;
    _size = pSize;

    std::cout << "BigString(char* pArrayOfChars, int pSize)" << std::endl;
}

BigString::BigString(const BigString& bs)
{
    shallowCopy(bs);
    std::cout << "BigString(const BigString& bs)" << std::endl;
}
```

2.



```cpp
BigString::~BigString()
{
    delete[] _arrayOfChars;
    std::cout << "~BigString()" << std::endl;
}
```

3.



```cpp
BigString& BigString::operator=(const BigString& bs)
{
    shallowCopy(bs);

    std::cout << "operator=(const BigString& bs)" << std::endl;

    return *this;
}
```

4.

```cpp
std::ostream& operator<<(std::ostream& out, BigString& bs)
{
    for (int i = 0; i < bs.getSize(); ++i)
    {
        out << bs[i];
    }
    return out;
}
```

```cpp
std::istream& operator>>(std::istream& in, BigString& bs)
{
    char inArray[512];
    in.getline(inArray, 512);

    const int inSize = in.gcount() - 1;

    char* arrayPtr = new char[inSize];

    for (int i = 0; i < inSize; i++)
    {
        arrayPtr[i] = inArray[i];
    }

    bs.setArrayOfChars(arrayPtr);
    bs.setSize(inSize);

    return in;
}
```

5.

```cpp
char& operator [] (int index) { return _arrayOfChars[index]; }
```

Test data:

```cpp
char* tp = new char[] {"test"};
```

Sample output:

```cpp
BigString big(tp, 4);
```

=

```
BigString(char* pArrayOfChars, int pSize)
~BigString()
```

```cpp
BigString bigger(big);
```

=

```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
~BigString()
~BigString()
```

```cpp
BigString biggest = bigger;
```

=

```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
BigString(const BigString& bs)
~BigString()
~BigString()
~BigString()
```

```cpp
BigString streamedIn;

std::cin >> streamedIn;

std::cout << streamedIn << std::endl;
```

=

```
abcdefghijkl
abcdefghijkl
~BigString()
```

Reflection:

I don’t think I really understand destructors that well, I was having a problem with mine but I eventually fixed it. I’m unsure when to use shallow copies or deep copies, deep copies seem superior

Metadata:

Gang of three

Further information:

When would you want to use shallow copies instead of deep copies?

Q2. Test harness

Question:

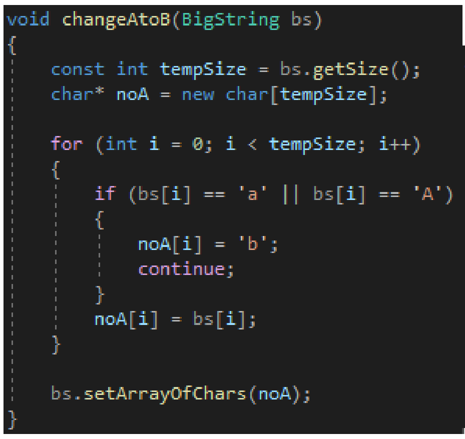
Create code to test all the functionality within BigString. Include at least the following tests:

1. Pass BigString to a function, by value
2. Pass BigString to a function, by reference
3. Return BigString from a function, by value
4. Return BigString from a function, by reference
5. Assign one BigString object to another

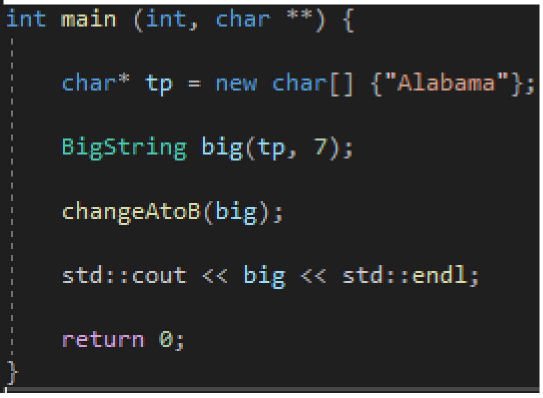
Also check for possible memory leaks

Solution:

1.

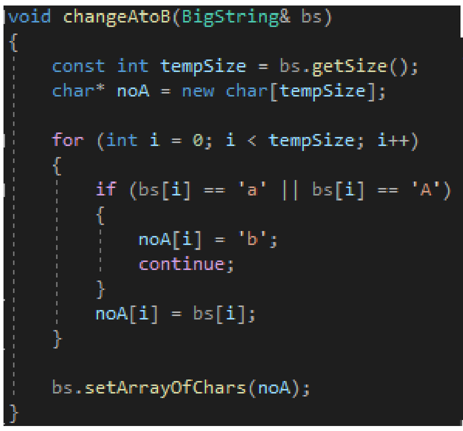


```
void changeAtoB(BigString bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);
}
```

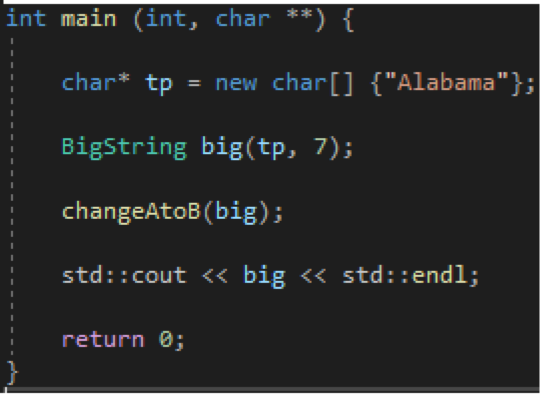


```
int main (int, char **) {

    char* tp = new char[] {"Alabama"};

    BigString big(tp, 7);

    changeAtoB(big);

    std::cout << big << std::endl;

    return 0;
}
```

2.

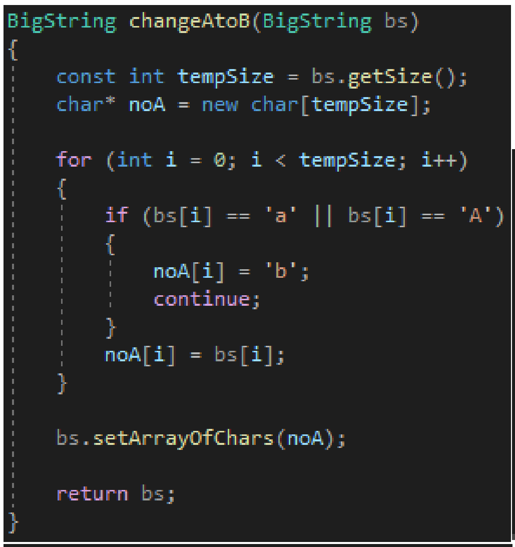


```
void changeAtoB(BigString& bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);
}
```

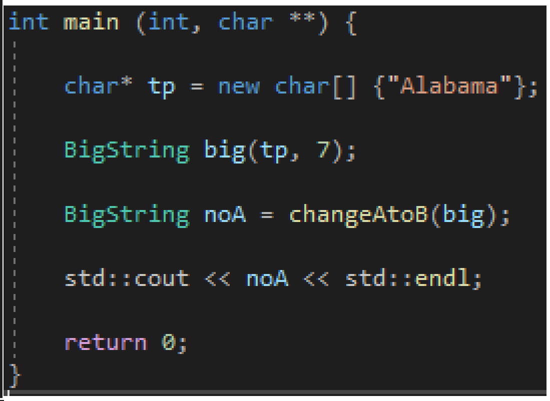


```
int main (int, char **) {

    char* tp = new char[] {"Alabama"};

    BigString big(tp, 7);

    changeAtoB(big);

    std::cout << big << std::endl;

    return 0;
}
```

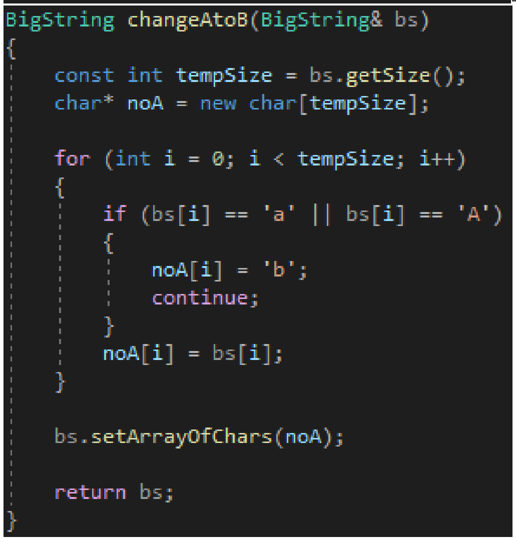
3.



```cpp
BigString changeAtoB(BigString bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);

    return bs;
}
```

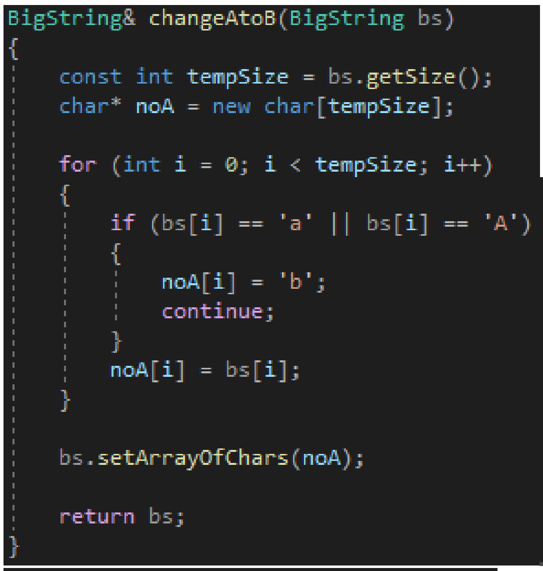


```cpp
int main (int, char **) {

    char* tp = new char[] {"Alabama"};

    BigString big(tp, 7);

    BigString noA = changeAtoB(big);

    std::cout << noA << std::endl;

    return 0;
}
```

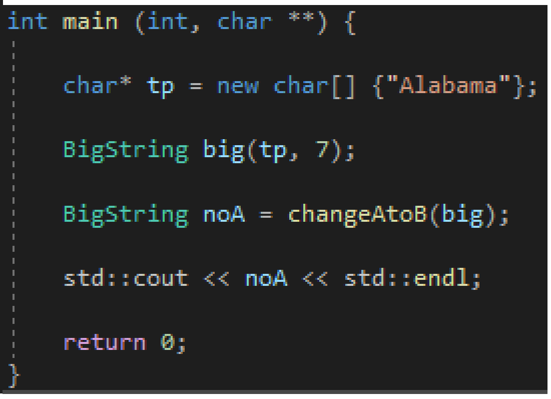


```cpp
BigString changeAtoB(BigString& bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);

    return bs;
}
```

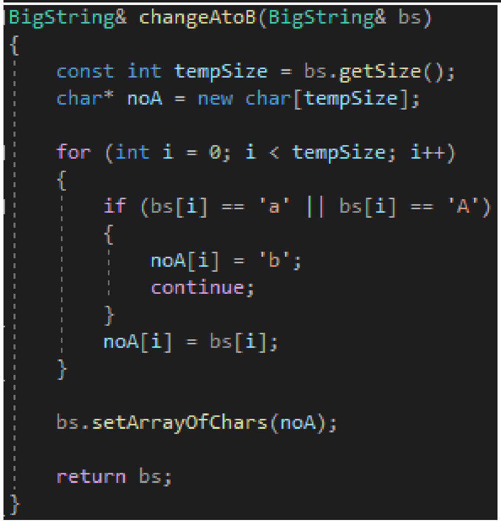
4.



```
BigString& changeAtoB(BigString bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);

    return bs;
}
```

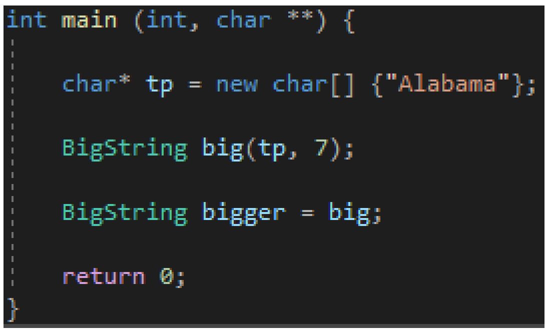


```
int main (int, char **) {

    char* tp = new char[] {"Alabama"};

    BigString big(tp, 7);

    BigString noA = changeAtoB(big);

    std::cout << noA << std::endl;

    return 0;
}
```



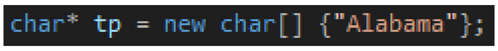
```
BigString& changeAtoB(BigString& bs)
{
    const int tempSize = bs.getSize();
    char* noA = new char[tempSize];

    for (int i = 0; i < tempSize; i++)
    {
        if (bs[i] == 'a' || bs[i] == 'A')
        {
            noA[i] = 'b';
            continue;
        }
        noA[i] = bs[i];
    }

    bs.setArrayOfChars(noA);

    return bs;
}
```

5.



```
int main (int, char **) {

    char* tp = new char[] {"Alabama"};

    BigString big(tp, 7);

    BigString bigger = big;

    return 0;
}
```

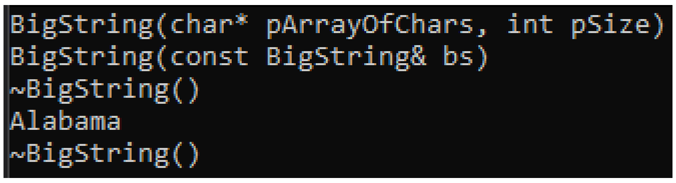
Test data:



```
char* tp = new char[] {"Alabama"};
```

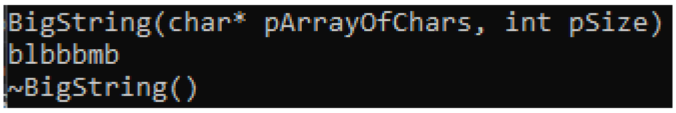
Sample output:

1.



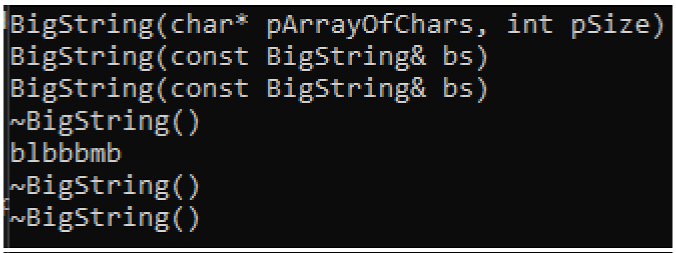
```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
~BigString()
Alabama
~BigString()
```

2.

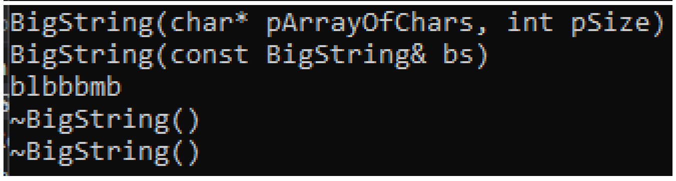


```
BigString(char* pArrayOfChars, int pSize)
blbbbmb
~BigString()
```

3.

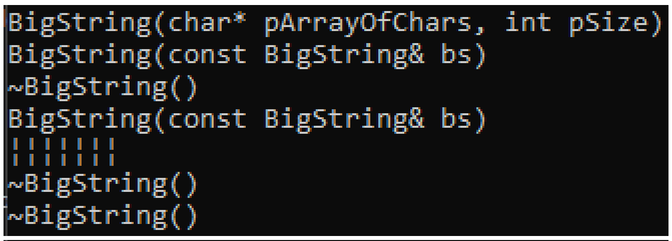


```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
BigString(const BigString& bs)
~BigString()
blbbbmb
~BigString()
~BigString()
```

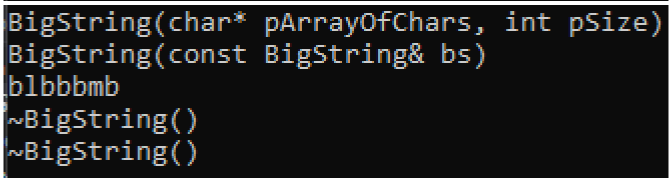


```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
blbbbmb
~BigString()
~BigString()
```

4.

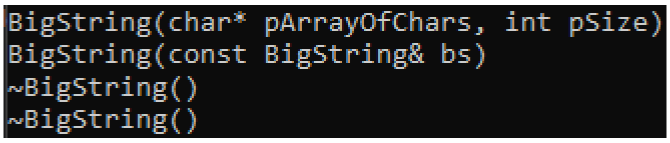


```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
~BigString()
BigString(const BigString& bs)
¦¦¦¦¦¦¦
~BigString()
~BigString()
```



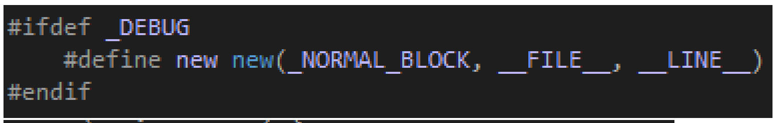
```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
blbbbmb
~BigString()
~BigString()
```

5.

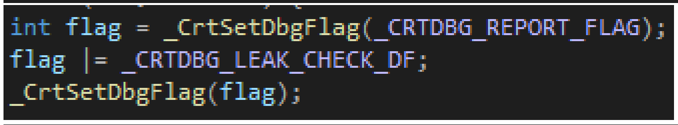


```
BigString(char* pArrayOfChars, int pSize)
BigString(const BigString& bs)
~BigString()
~BigString()
```

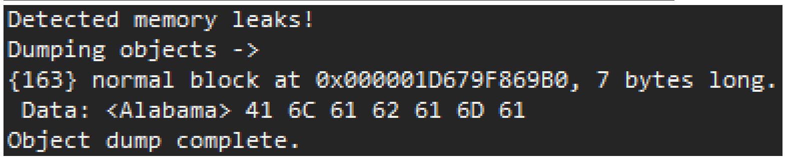
Memory Leak Detecting



```
#ifdef _DEBUG
    #define new new(_NORMAL_BLOCK, __FILE__, __LINE__)
#endif
```



```
int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
flag |= _CRTDBG_LEAK_CHECK_DF;
_CrtSetDbgFlag(flag);
```



```
Detected memory leaks!
Dumping objects ->
{163} normal block at 0x000001D679F869B0, 7 bytes long.
 Data: <Alabama> 41 6C 61 62 61 6D 61
Object dump complete.
```

Reflection:

I'm not sure what part 5 is really expecting, isn't it just the same as what I have done for the assignment operator testing? Is it to make us think about shallow and deep copies?

I also struggle to understand detecting memory leeks, I think I understand what a memory leek is (solution 1 of part 4 is an example of a memory leek) but the detecting part is confusing to me, **I was doing it right I was just looking in the wrong place**.

Metadata:

Testing

Further information:

Can you go through memory leek testing briefly?

Q3. Optimisation

Question:

Now that you have a clear picture of which BigString functions are being called; are there any situations where you think you can improve the performance of your code?

Solution:

n/a

Test data:

n/a

Sample output:

n/a

Reflection:

The only optimisation I can really think of is maybe using uniform initialization for the default constructor, I'm not sure if this actually has an effect on performance though. We could potentially overload the new and delete operators as well but I don't understand this enough to be able to say if it would improve performance

Metadata:

Optimise

Further information:

Are there ways to optimise

Week 7 – Lab G

Date: 15/11/2022

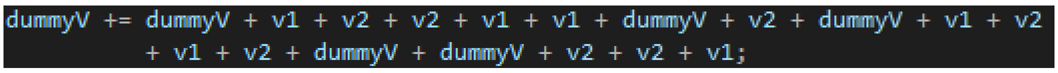
Q1. Benchmarks

Question:

Produce a set of reliable, reproducible and effective benchmarks.

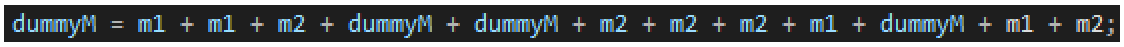
Solution:

Vector3d Benchmark



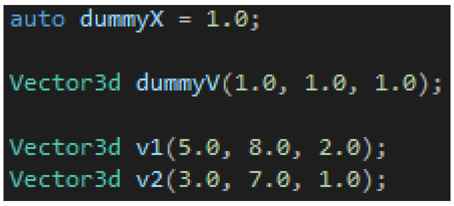
```
dummyV += dummyV + v1 + v2 + v2 + v1 + v1 + dummyV + v2 + dummyV + v1 + v2
          + v1 + v2 + dummyV + dummyV + v2 + v2 + v1;
```

Matrix33d Benchmark

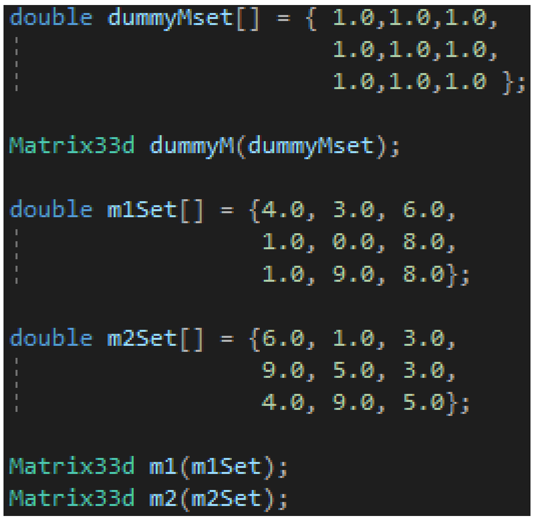


```
dummyM = m1 + m1 + m2 + dummyM + dummyM + m2 + m2 + m2 + m1 + dummyM + m1 + m2;
```

Test data:



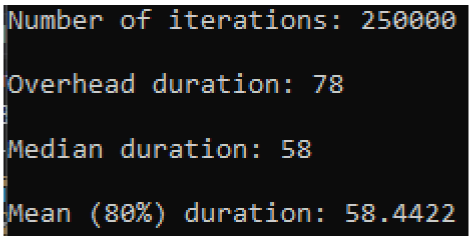
```
auto dummyX = 1.0;

Vector3d dummyV(1.0, 1.0, 1.0);

Vector3d v1(5.0, 8.0, 2.0);
Vector3d v2(3.0, 7.0, 1.0);
```



```
double dummyMset[] = { 1.0,1.0,1.0,
                       1.0,1.0,1.0,
                       1.0,1.0,1.0 };

Matrix33d dummyM(dummyMset);

double m1Set[] = {4.0, 3.0, 6.0,
                  1.0, 0.0, 8.0,
                  1.0, 9.0, 8.0};

double m2Set[] = {6.0, 1.0, 3.0,
                  9.0, 5.0, 3.0,
                  4.0, 9.0, 5.0};

Matrix33d m1(m1Set);
Matrix33d m2(m2Set);
```

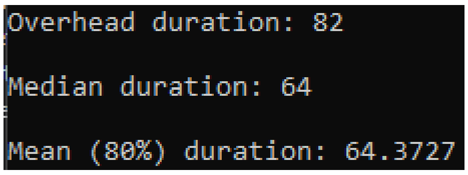
Sample output:

Vector3d Benchmark Results



```
Number of iterations: 250000

Overhead duration: 78

Median duration: 58

Mean (80%) duration: 58.4422
```

Matrix33d Benchmark Results



```
Overhead duration: 82

Median duration: 64

Mean (80%) duration: 64.3727
```

Reflection:

There isn't much to comment on with this task. The Vector3d class took a bit more to get the timing up to a suitable level for a benchmark I assume that's just because the matrix maths is more expensive.

Metadata:

Benchmarks

Further information:

n/a

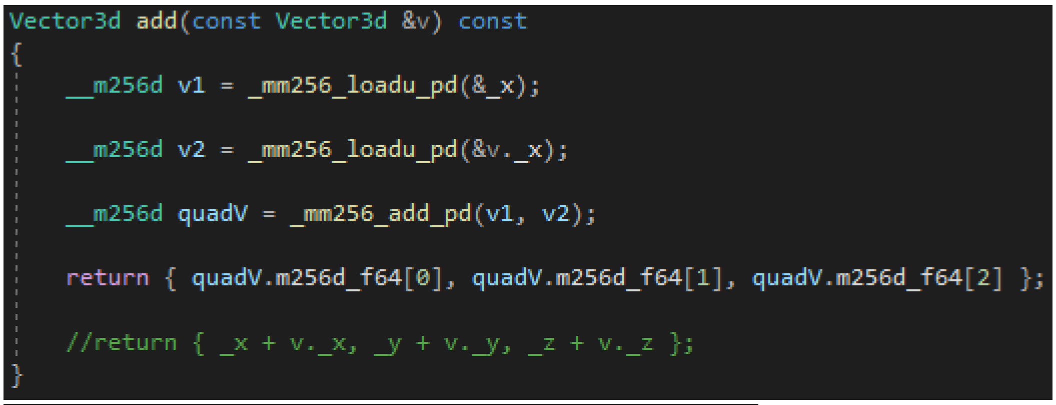
Q2. SIMD

Question:

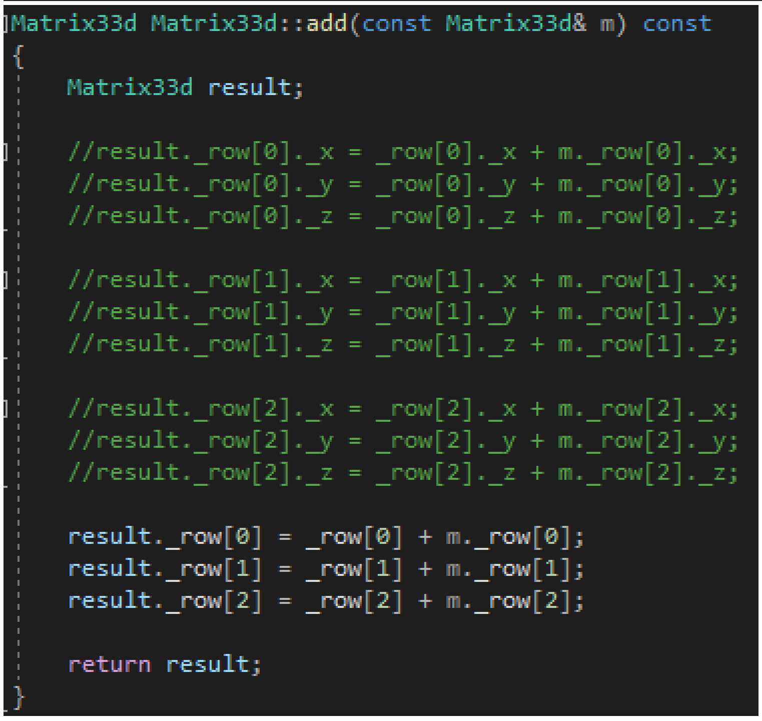
Convert your vector and matrix classes to use the SIMD instructions. Time your results. How did they compare to your original implementation?

Solution:

Vector SIMD



```
Vector3d add(const Vector3d &v) const
{
    __m256d v1 = _mm256_loadu_pd(&_x);

    __m256d v2 = _mm256_loadu_pd(&v._x);

    __m256d quadV = _mm256_add_pd(v1, v2);

    return { quadV.m256d_f64[0], quadV.m256d_f64[1], quadV.m256d_f64[2] };

    //return { _x + v._x, _y + v._y, _z + v._z };
}
```



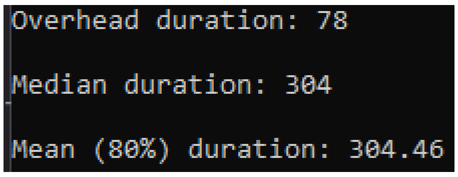
```
Matrix33d Matrix33d::add(const Matrix33d& m) const
{
    Matrix33d result;

    //result._row[0]._x = _row[0]._x + m._row[0]._x;
    //result._row[0]._y = _row[0]._y + m._row[0]._y;
    //result._row[0]._z = _row[0]._z + m._row[0]._z;

    //result._row[1]._x = _row[1]._x + m._row[1]._x;
    //result._row[1]._y = _row[1]._y + m._row[1]._y;
    //result._row[1]._z = _row[1]._z + m._row[1]._z;

    //result._row[2]._x = _row[2]._x + m._row[2]._x;
    //result._row[2]._y = _row[2]._y + m._row[2]._y;
    //result._row[2]._z = _row[2]._z + m._row[2]._z;

    result._row[0] = _row[0] + m._row[0];
    result._row[1] = _row[1] + m._row[1];
    result._row[2] = _row[2] + m._row[2];

    return result;
}
```

Test data:

Previously outlined benchmarks

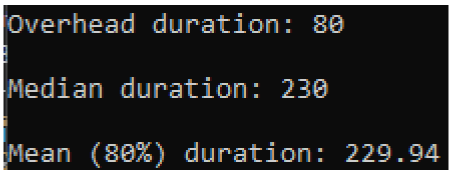
Sample output:

Vector SIMD Timing



```
Overhead duration: 78

Median duration: 304

Mean (80%) duration: 304.46
```

Matrix SIMD Timing



```
Overhead duration: 80

Median duration: 230

Mean (80%) duration: 229.94
```

Reflection:

I must be doing something wrong because instead of getting quicker each got 4-6 times slower. Maybe I'm not targeting the right CPU architecture? Although I did double check that I was.

Metadata:

SIMD Intrinsic

Further information:

What am I doing wrong? Is it that I'm not targeting the correct architecture or something? **It is expected that you implementation will be slower, the loading in your solution is bad as it relies on x, y, z to be next to each other in memory.**

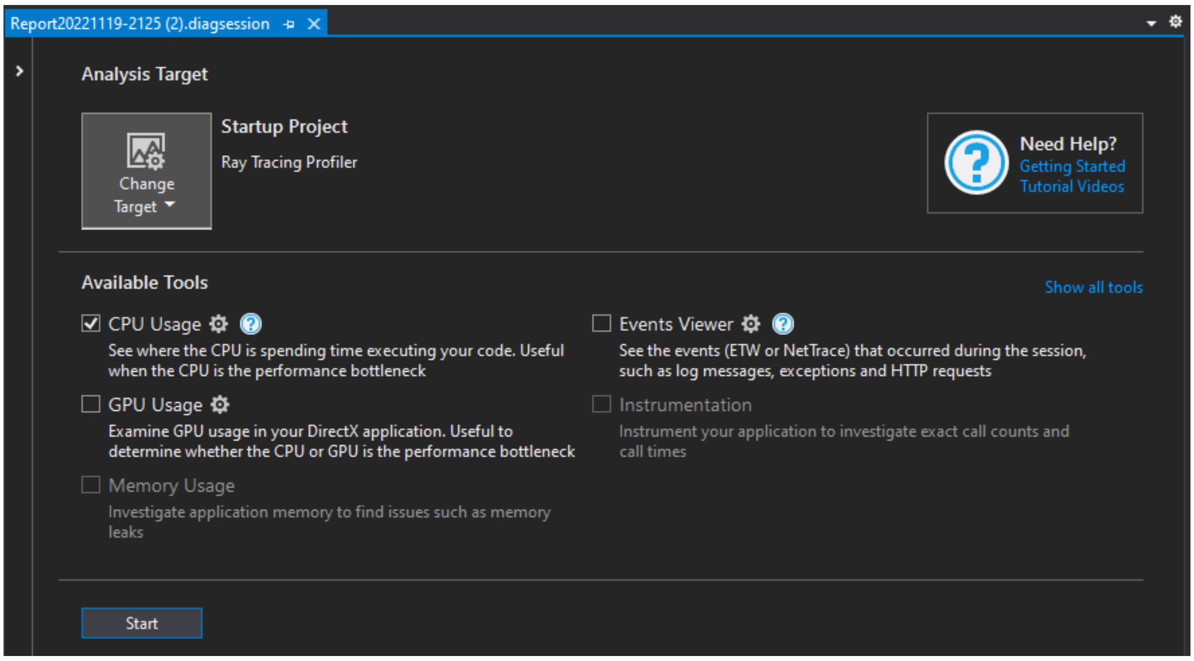
Q3. Profilers

Question:

Profile the Raytracer, using both a sampling and instrumented profiler.

Solution:

Sampling



Report20221119-2125 (2).diagsession

Analysis Target

Startup Project

Ray Tracing Profiler

Need Help?
Getting Started
Tutorial Videos

Available Tools

Show all tools

[✓] CPU Usage
See where the CPU is spending time executing your code. Useful when the CPU is the performance bottleneck

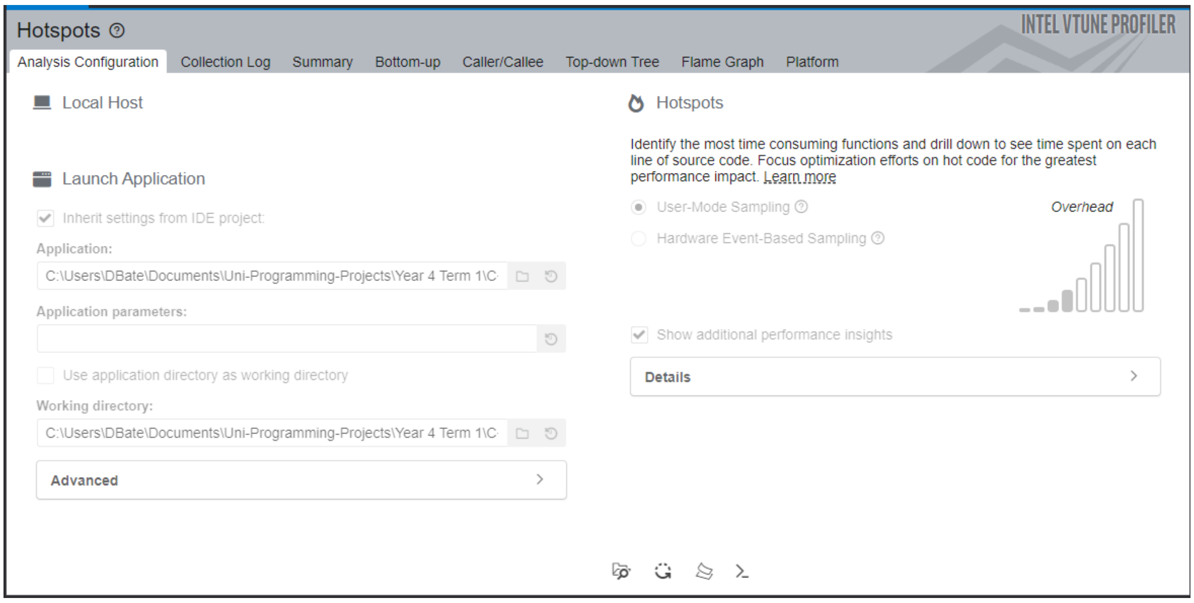
[ ] Events Viewer
See the events (ETW or NetTrace) that occurred during the session, such as log messages, exceptions and HTTP requests

[ ] GPU Usage
Examine GPU usage in your DirectX application. Useful to determine whether the CPU or GPU is the performance bottleneck

[ ] Instrumentation
Instrument your application to investigate exact call counts and call times

[ ] Memory Usage
Investigate application memory to find issues such as memory leaks

Instrumented



Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform

INTEL VTUNE PROFILER

Local Host

Launch Application

[✓] Inherit settings from IDE project

Application:
C:\Users\DBate\Documents\Uni-Programming-Projects\Year 4 Term 1\C

Application parameters:

[ ] Use application directory as working directory

Working directory:
C:\Users\DBate\Documents\Uni-Programming-Projects\Year 4 Term 1\C

Advanced

Hotspots

Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. Learn more

● User-Mode Sampling
○ Hardware Event-Based Sampling

[✓] Show additional performance insights

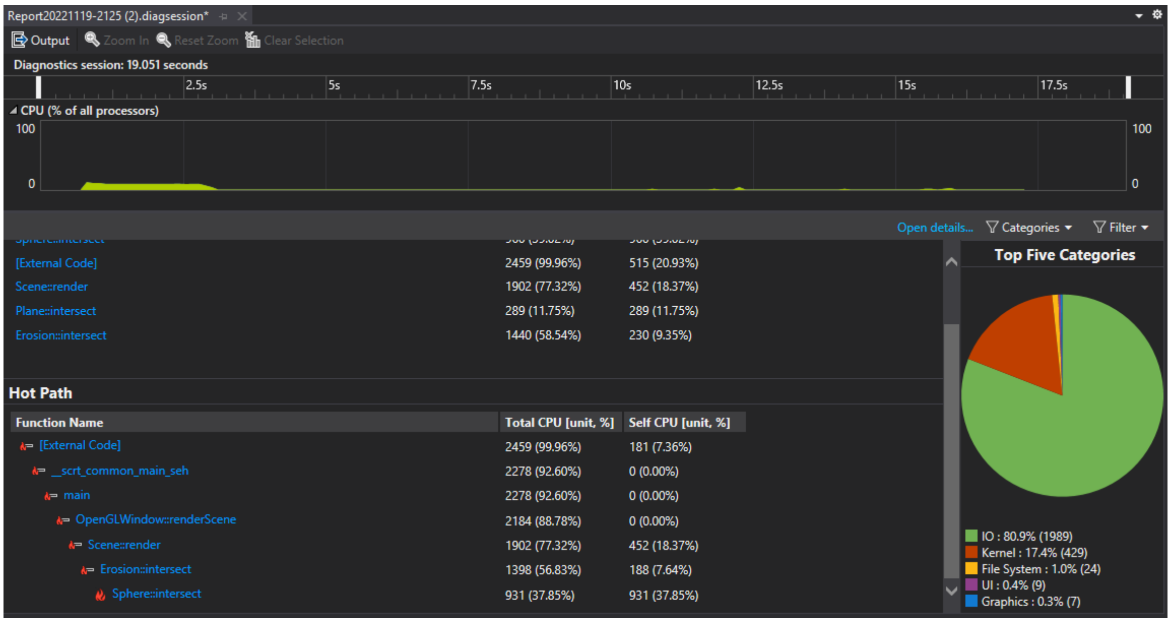
Details

Test data:

n/a

Sample output:

Sampling



Report20221119-2125 (2).diagsession*

Output | Zoom In | Reset Zoom | Clear Selection

Diagnostics session: 19.051 seconds

CPU (% of all processors)

[External Code]	2459 (99.96%)	515 (20.93%)
Scene::render	1902 (77.32%)	452 (18.37%)
Plane::intersect	289 (11.75%)	289 (11.75%)
Erosion::intersect	1440 (58.54%)	230 (9.35%)

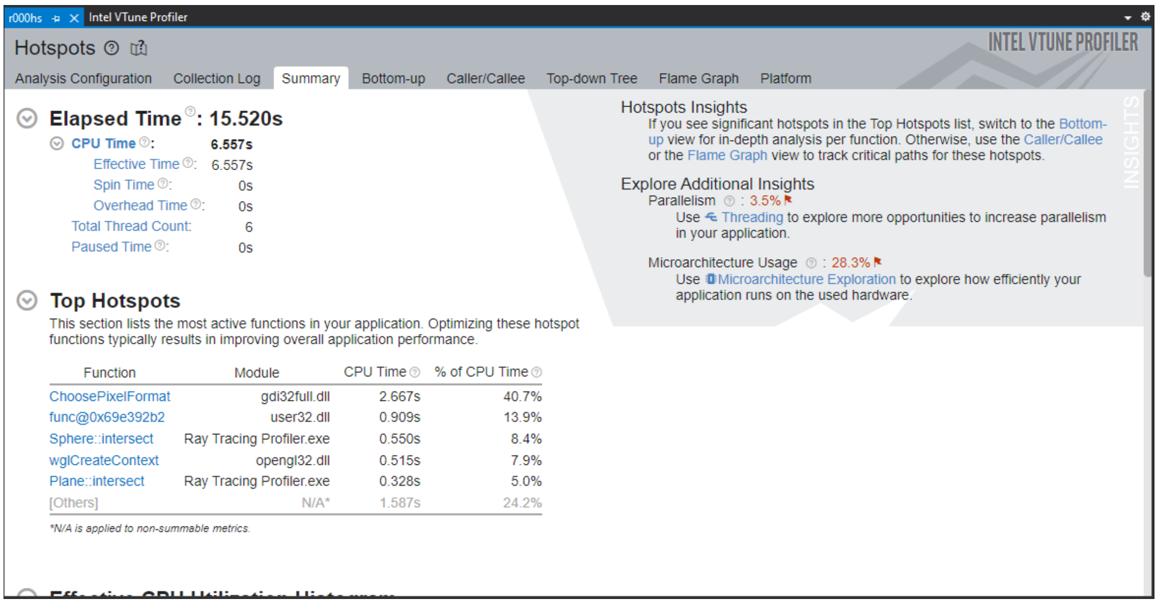
Hot Path

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
[External Code]	2459 (99.96%)	181 (7.36%)
__scrt_common_main_seh	2278 (92.60%)	0 (0.00%)
main	2278 (92.60%)	0 (0.00%)
OpenGLWindow::renderScene	2184 (88.78%)	0 (0.00%)
Scene::render	1902 (77.32%)	452 (18.37%)
Erosion::intersect	1398 (56.83%)	188 (7.64%)
Sphere::intersect	931 (37.85%)	931 (37.85%)

Top Five Categories

- 🟩 IO : 80.9% (1989)
- 🟥 Kernel : 17.4% (429)
- 🟨 File System : 1.0% (24)
- 🟪 UI : 0.4% (9)
- 🟦 Graphics : 0.3% (7)

Instrumented



r000hs | Intel VTune Profiler

Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform

Elapsed Time: 15.520s

CPU Time:	6.557s
Effective Time:	6.557s
Spin Time:	0s
Overhead Time:	0s
Total Thread Count:	6
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
ChoosePixelFormat	gdi32full.dll	2.667s	40.7%
func@0x69e392b2	user32.dll	0.909s	13.9%
Sphere::intersect	Ray Tracing Profiler.exe	0.550s	8.4%
wglCreateContext	opengl32.dll	0.515s	7.9%
Plane::intersect	Ray Tracing Profiler.exe	0.328s	5.0%
[Others]	N/A*	1.587s	24.2%

*N/A is applied to non-summable metrics.

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism : 3.5%

Use Threading to explore more opportunities to increase parallelism in your application.

Microarchitecture Usage : 28.3%

Use Microarchitecture Exploration to explore how efficiently your application runs on the used hardware.

Reflection:

This was easy to do once I got VTune installed and found the performance profiler, it was in a different place than described in the video **Debug -> Performance Profiler.** I think I prefer the Microsoft profiler better, something about it is easier to read for me. Although intel definitely gives more information.

Metadata:

Profiling

Further information:

Which is better instrumentation or sampling? It depends on what you are after instrumentation is more detailed

Final Lab

Graphics (700106)

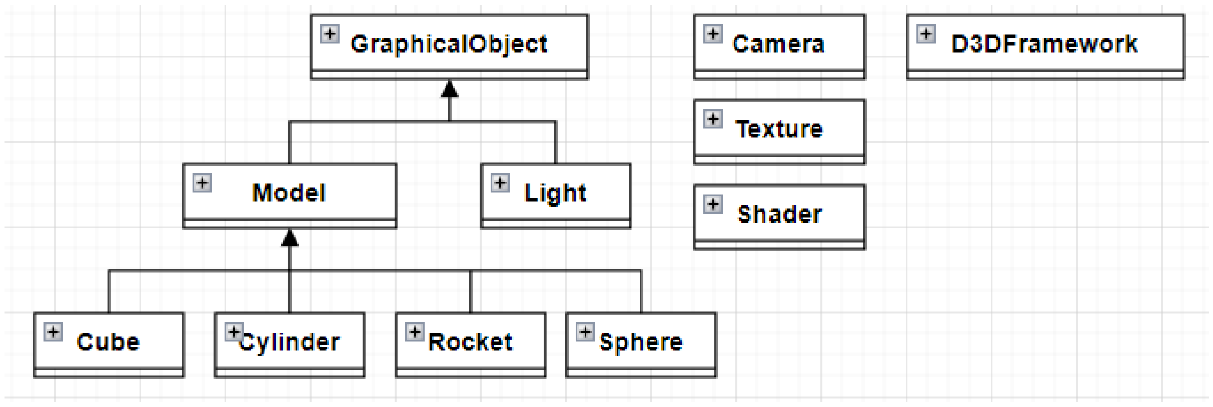
For the project I needed 4 main models, a cube, a cylinder, a rocket and a sphere, so I made these models in blender and exported them into obj files. Then in the project I loaded the geometry data using assimp and a loadOBJ method I created. This method takes in a file name, a list of simple vertices and a list of indices and then loads the appropriate file's vertices, normals and texture coordinates into then list of simple vertices, and then loads the files indices into the lists of indices. Once all the files have been loaded into the vertex and index lists these lists are used to create the vertex and index buffers. I stored all of the geometry data into a single vertex buffer which was easier to do but I made for draw objects in the scene a bit harder as I had to know where each objects data was in the vertex buffer, if I were to do this again I would look into using separate vertex buffers for each of the models.

The shading and lighting are all handled by 1 vertex and 1 pixel shader. The shader has the world, view and projection passed in by the constant buffer, the constant buffer also passes in the light positions and colours. The texture and sampler of an object are passed directly into the pixel shader. Using this information, it is determined what colour a pixel should be based on the ambient, diffuse and specular lighting calculation.

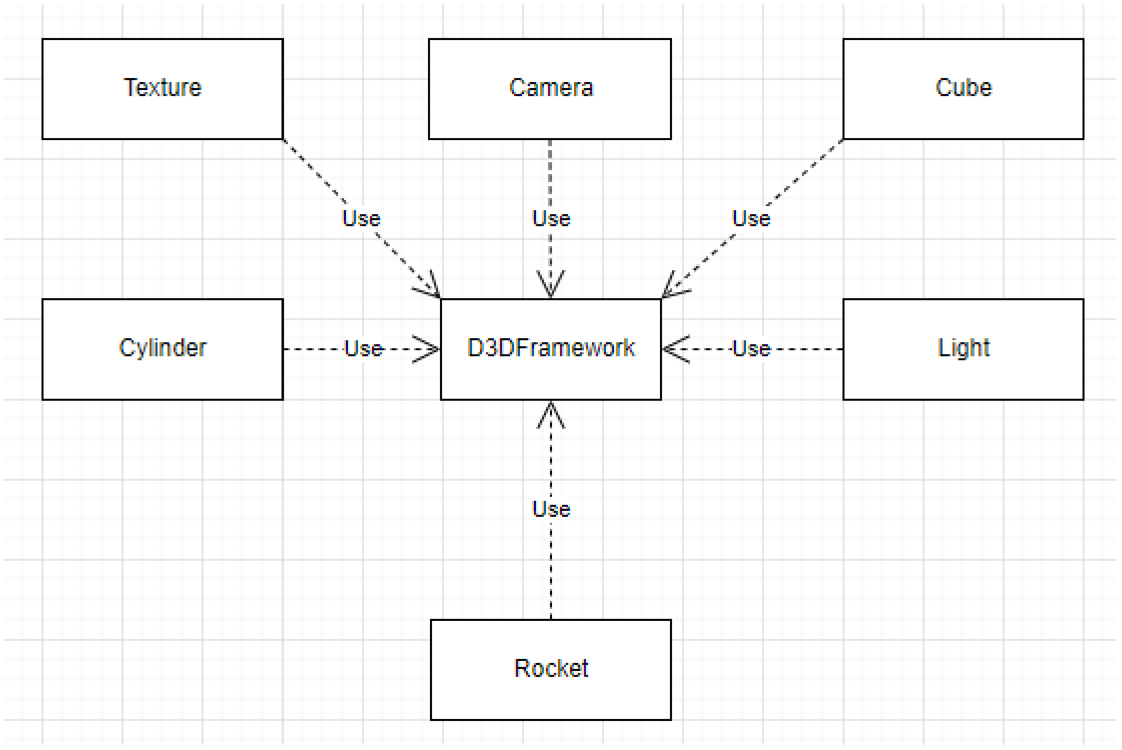
The project could potentially be extended with the addition of the rocket engine and the particle system that accompanies it, some additional shaders that give the simulation a different look, for example a toon shader. Being able to add in the terrain deformation would have also helped to make the simulation more visually appealing. The program code also benefit from the addition of a configuration file and parser to allow some of the code to moved out of the main program into a config file. Shadows would also help to improve the overall look of the simulation especially regarding the lighting. I would have also liked to add some fluid simulation, either in addition or instead of the terrain deformation, I think this would be an interesting challenge while also adding a lot to the simulation in terms of visual appeal. I also would have like to add a physical sun and moon into the simulation like I had planned, this way you would be able to see the sun and moon rising and setting in the scene.

Design (700120)

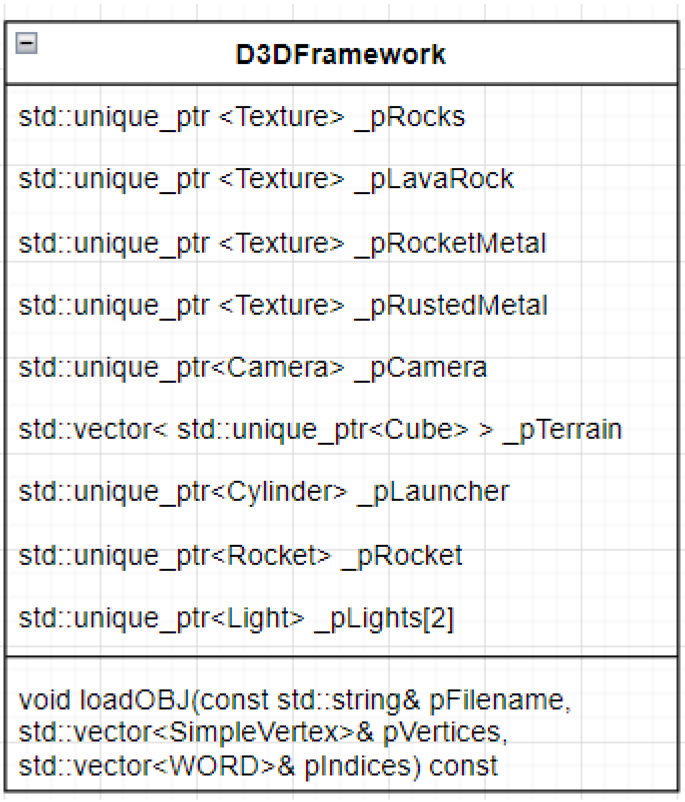
The main classes of the project are; D3DFramework, Camera, Texture, Shader, GraphicalObject, Model, Light, Cube, Cylinder, Rocket and Sphere. Each of these classes serve a unique purpose in the project. The class structure of each of these is as follows:



And these classes interact with each other as shown below:



The D3DFramework class serves as the main class responsible for running the simulation. On top of the member variables listed below the class contains all the buffer, device, context, swapChain, vertex and pixel shader pointers. The main additions I made to this class were the unique pointers that I needed for the simulation and the loadOBJ method. There are some weaknesses with the design, for example, it would have probably been better to sort the Textures in a list of unique pointers to texture, but I found this was eaiser to manage in the cpp as it was clearer what texture was being used where.

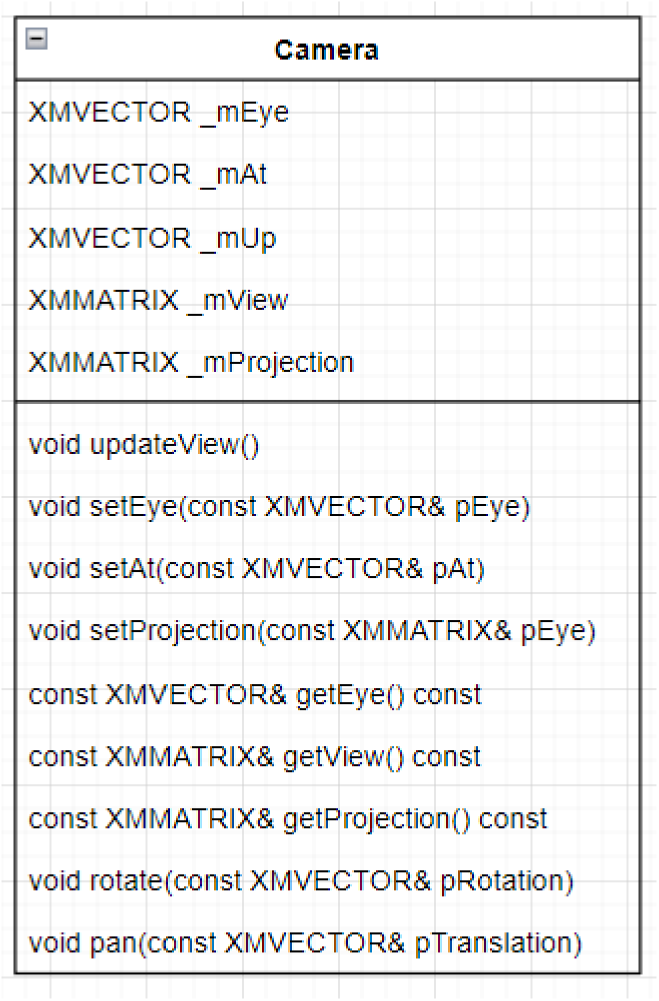


D3DFramework

std::unique_ptr <Texture> _pRocks
std::unique_ptr <Texture> _pLavaRock
std::unique_ptr <Texture> _pRocketMetal
std::unique_ptr <Texture> _pRustedMetal
std::unique_ptr<Camera> _pCamera
std::vector< std::unique_ptr<Cube> > _pTerrain
std::unique_ptr<Cylinder> _pLauncher
std::unique_ptr<Rocket> _pRocket
std::unique_ptr<Light> _pLights[2]

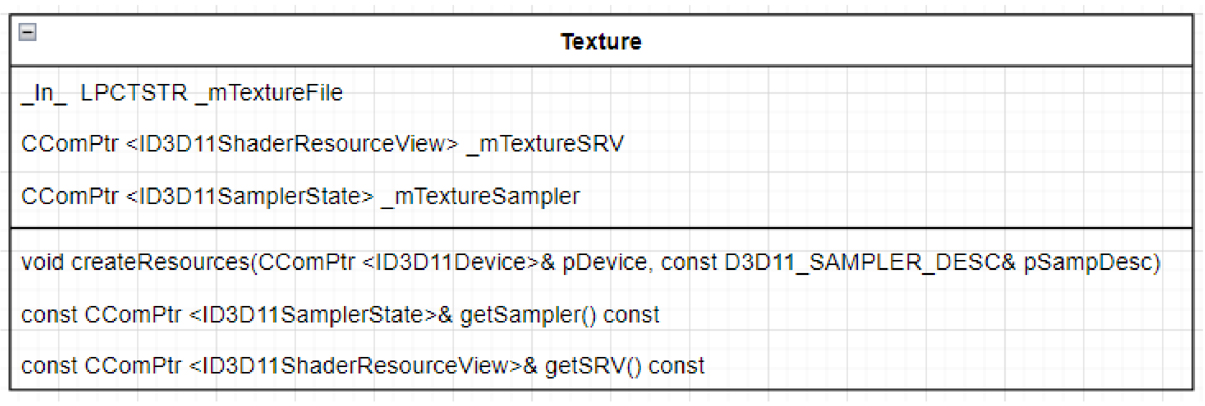
void loadOBJ(const std::string& pFilename, std::vector<SimpleVertex>& pVertices, std::vector<WORD>& pIndices) const

The camera class is responsible for storing the variables that represent the view that the user will have of the simulation at runtime. It is a pretty simple class as it has very limited functionality, it mainly consists of getters and setters as well as a couple of methods that allow for manipulation of the camera view. I think the only real weakness in this design is the setProjection() method, it would have probably been better to pass in the project at the construction of the camera since the projection does not need to be changed after construction.



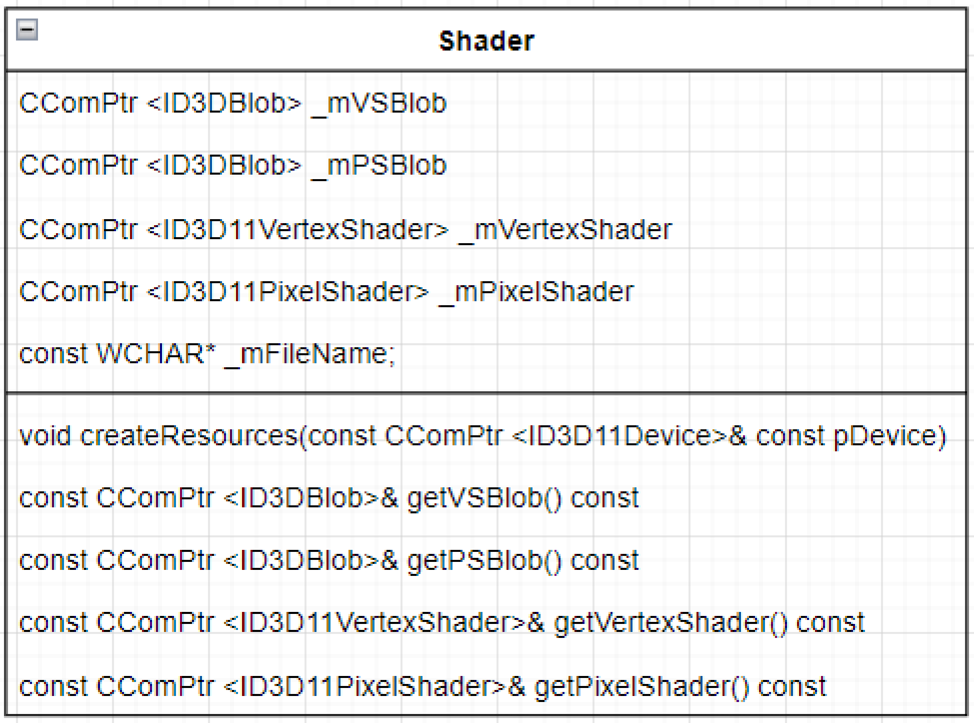
```
Camera
XMVECTOR _mEye
XMVECTOR _mAt
XMVECTOR _mUp
XMMATRIX _mView
XMMATRIX _mProjection

void updateView()
void setEye(const XMVECTOR& pEye)
void setAt(const XMVECTOR& pAt)
void setProjection(const XMMATRIX& pEye)
const XMVECTOR& getEye() const
const XMMATRIX& getView() const
const XMMATRIX& getProjection() const
void rotate(const XMVECTOR& pRotation)
void pan(const XMVECTOR& pTranslation)
```

The Texture class is responsible for loading a given texture into a shader resource view and a sampler state so that it can be referenced as needed. I can’t really spot any major weakness in this design as it is very simple. It was made to reduce the repetition of the code responsible for loading texture shader resource view and sampler states, so it serves its purpose quite well.



```
Texture
_In_ LPCTSTR _mTextureFile
CComPtr <ID3D11ShaderResourceView> _mTextureSRV
CComPtr <ID3D11SamplerState> _mTextureSampler

void createResources(CComPtr <ID3D11Device>& pDevice, const D3D11_SAMPLER_DESC& pSampDesc)
const CComPtr <ID3D11SamplerState>& getSampler() const
const CComPtr <ID3D11ShaderResourceView>& getSRV() const
```

The shader class is not actually used in the project as I couldn't get it to work properly and since I only use one shader it was not worth spending the time to fix it. This design also has a few flaws the main being that this class doesn't allow you to have a separate vertex and pixel shaders, this design pairs them together.



Shader

CComPtr <ID3DBlob> _mVSBlob

CComPtr <ID3DBlob> _mPSBlob

CComPtr <ID3D11VertexShader> _mVertexShader

CComPtr <ID3D11PixelShader> _mPixelShader

const WCHAR* _mFileName;

void createResources(const CComPtr <ID3D11Device>& const pDevice)

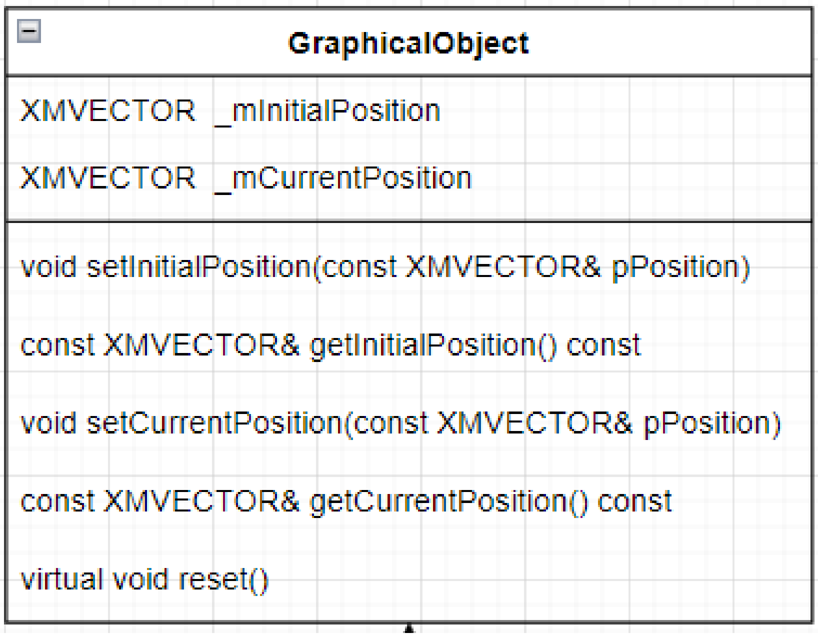
const CComPtr <ID3DBlob>& getVSBlob() const

const CComPtr <ID3DBlob>& getPSBlob() const

const CComPtr <ID3D11VertexShader>& getVertexShader() const

const CComPtr <ID3D11PixelShader>& getPixelShader() const

The GraphicalObject class serves as a base for other classes to inherit from, all it does is define that a graphical object must have an initial and current position. The class also outlines a virtual reset class that can be overridden but this definition makes the current position the same as the initial position. The setInitialPosition is set as protected which may not have been the best design but I thought it would make sense for the initial position to be changed outside of graphical object classes



GraphicalObject

XMVECTOR _mInitialPosition

XMVECTOR _mCurrentPosition

void setInitialPosition(const XMVECTOR& pPosition)

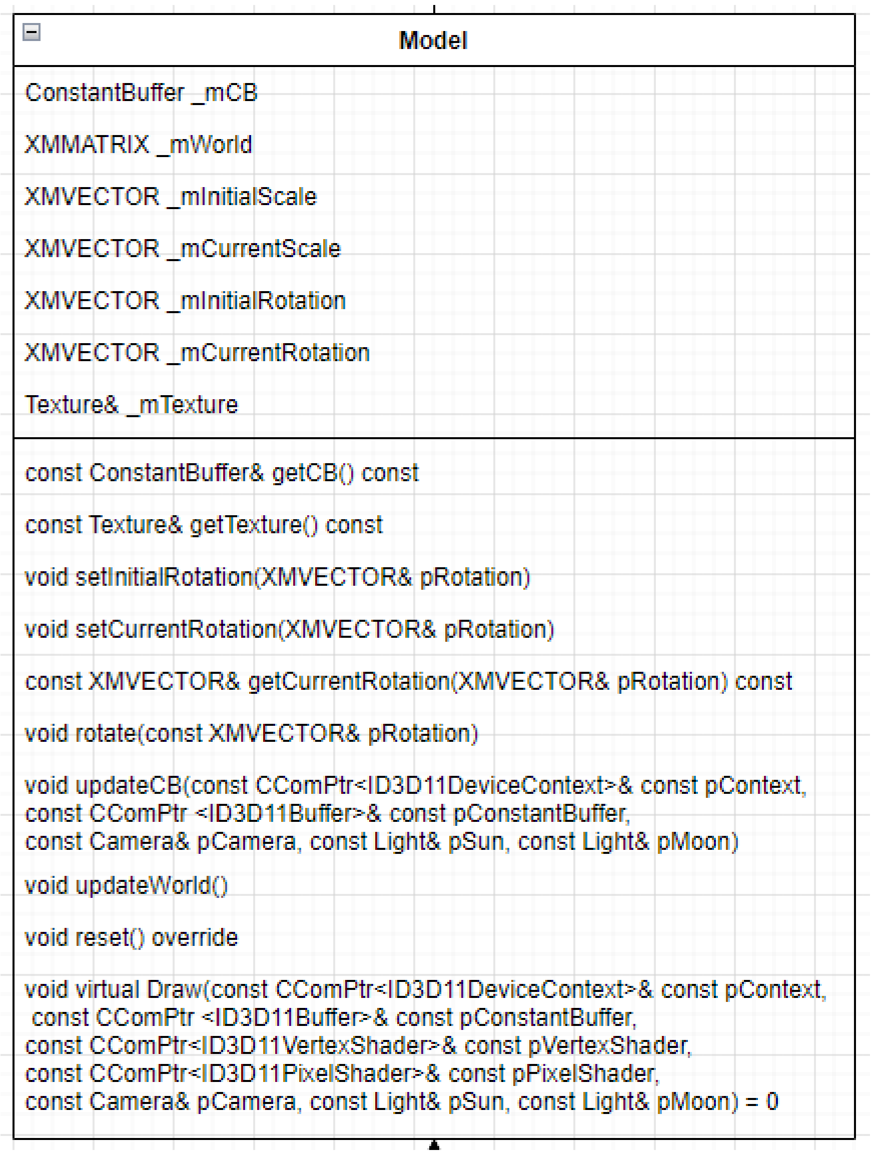
const XMVECTOR& getInitialPosition() const

void setCurrentPosition(const XMVECTOR& pPosition)

const XMVECTOR& getCurrentPosition() const

virtual void reset()

The model class serves the purpose of outlining some variables and methods for classes that will be drawable. it derives from the GraphicalObject class and expands upon it by adding a ConstantBuffer variable which is a struct defined within the model class, this struct is composed of a view, projection and world matrix as well as an array of light positions and light colours. The class also adds rotation and scale values, both initial and current, and a texture reference. The class adds getters for the texture reference, the constant buffer struct and current rotation, and setters for the initial and current rotation. Some of the most important methods added are draw, updateCB and updateWorld. The update world method redefines the object's world matrix to be the current transformation matrices multipled together. The updateCB method updates the object constant buffer using the object world, the camera's view and projection and the lights position and colour, once updated the object's constant buffer is applied to D3DFramework constant buffer ready to draw the model. Finally the draw method, this is a pure virtual function so its functionality is defined by its derived classes. The major weakness with the design is how many variables need to be passed into some of the methods but I couldn’t come up with a better solution.


Model

ConstantBuffer _mCB

XMMATRIX _mWorld

XMVECTOR _mInitialScale

XMVECTOR _mCurrentScale

XMVECTOR _mInitialRotation

XMVECTOR _mCurrentRotation

Texture& _mTexture

const ConstantBuffer& getCB() const

const Texture& getTexture() const

void setInitialRotation(XMVECTOR& pRotation)

void setCurrentRotation(XMVECTOR& pRotation)

const XMVECTOR& getCurrentRotation(XMVECTOR& pRotation) const

void rotate(const XMVECTOR& pRotation)

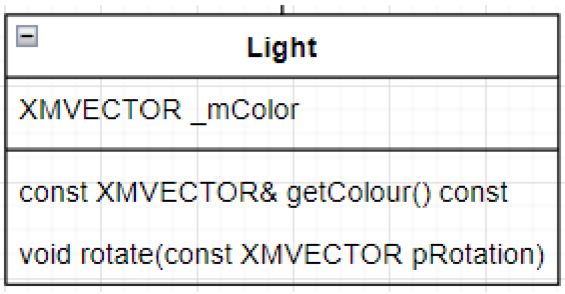
void updateCB(const CComPtr<ID3D11DeviceContext>& const pContext,
const CComPtr <ID3D11Buffer>& const pConstantBuffer,
const Camera& pCamera, const Light& pSun, const Light& pMoon)

void updateWorld()

void reset() override

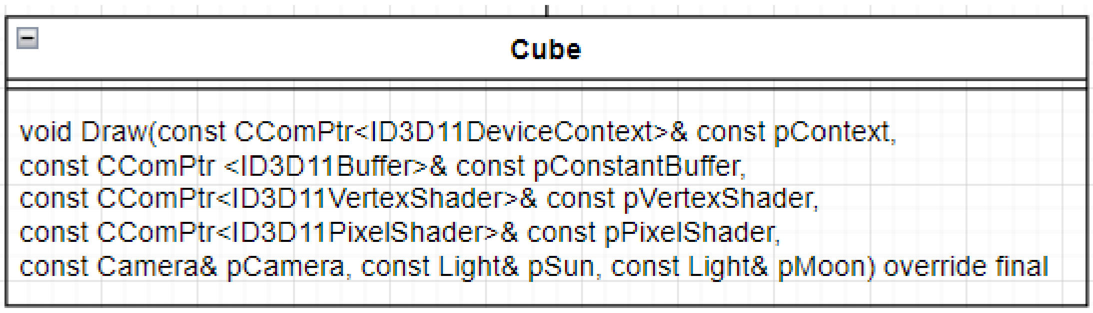
void virtual Draw(const CComPtr<ID3D11DeviceContext>& const pContext,
const CComPtr <ID3D11Buffer>& const pConstantBuffer,
const CComPtr<ID3D11VertexShader>& const pVertexShader,
const CComPtr<ID3D11PixelShader>& const pPixelShader,
const Camera& pCamera, const Light& pSun, const Light& pMoon) = 0

The light class serves the purpose of defining the lighting of a scene, it derives from the GraphicalObject class and expands upon it by adding a colour variable with an appropriate getter and a method that allows for the light to be rotated. This class does everything it needs to for this project but to make it more expandable I would want to add other methods to allow for more light transformations.



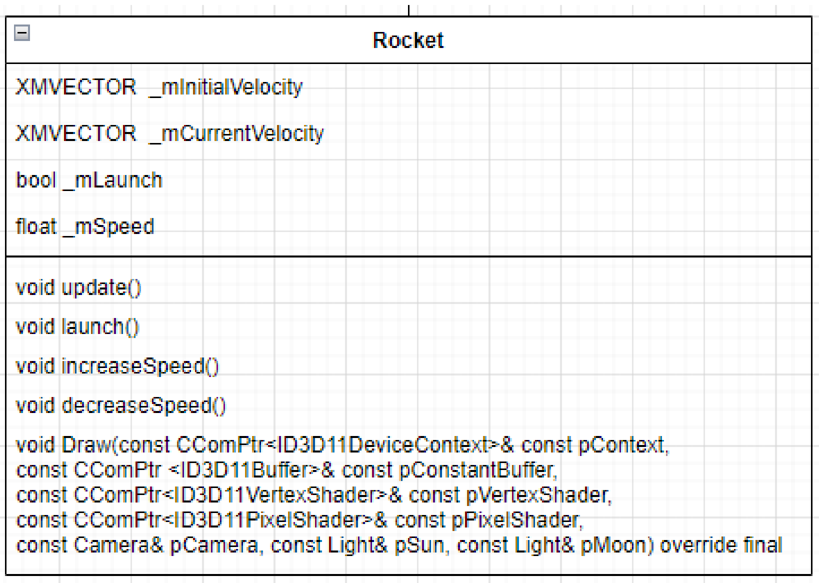
```
Light
XMVECTOR _mColor
const XMVECTOR& getColour() const
void rotate(const XMVECTOR pRotation)
```

The Cube, Cylinder and Sphere classes are all very similar so I'll talk about them all in this section, Rocket is also similar but has some extra functionality and will be discussed separately. These classes all derive from the model class and serve to override the draw method to apply to where each model's data is on the vertex and index buffers. This is the best way I could think to achieve my goals without having multiple vertex and index buffers.



```
Cube
void Draw(const CComPtr<ID3D11DeviceContext>& const pContext,
const CComPtr <ID3D11Buffer>& const pConstantBuffer,
const CComPtr<ID3D11VertexShader>& const pVertexShader,
const CComPtr<ID3D11PixelShader>& const pPixelShader,
const Camera& pCamera, const Light& pSun, const Light& pMoon) override final
```

The rocket class adds velocity and projectile motion. Some additional methods control the speed of the projectile motion as well as if the animation of the projectile motion should play. There are probably better was to do this, for example maybe the rotate method could have been in the model class, the speed contol methods probably could have been too if they had animations.



```
Rocket
XMVECTOR _mInitialVelocity
XMVECTOR _mCurrentVelocity
bool _mLaunch
float _mSpeed
void update()
void launch()
void increaseSpeed()
void decreaseSpeed()
void Draw(const CComPtr<ID3D11DeviceContext>& const pContext,
const CComPtr <ID3D11Buffer>& const pConstantBuffer,
const CComPtr<ID3D11VertexShader>& const pVertexShader,
const CComPtr<ID3D11PixelShader>& const pPixelShader,
const Camera& pCamera, const Light& pSun, const Light& pMoon) override final
```