<u>CSCI 522 – Game Engine - Fall 2024 – M2</u>

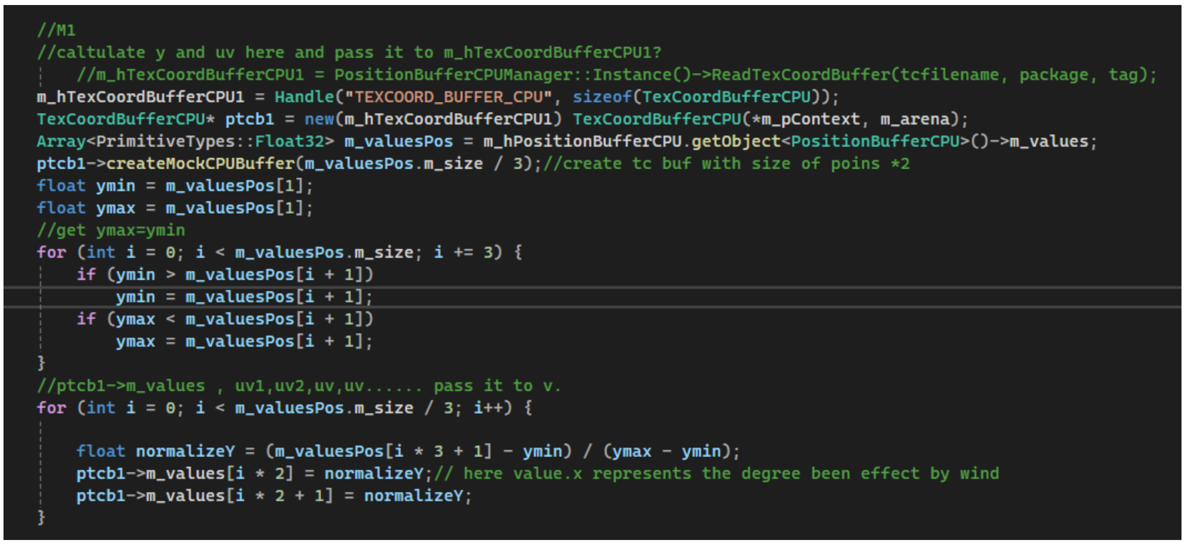
USC ID: 1154164561
NAME: Srija Madarapu
EMAIL: madarapu@usc.edu

Link to the video:

https://drive.google.com/file/d/1QuyjPleeNYMvAhMBNCSVtrTw9agTECd9/view?usp=sharing

M2: Creating wind effect by using solider positions as wind source and using a light source showing the light bending effect.

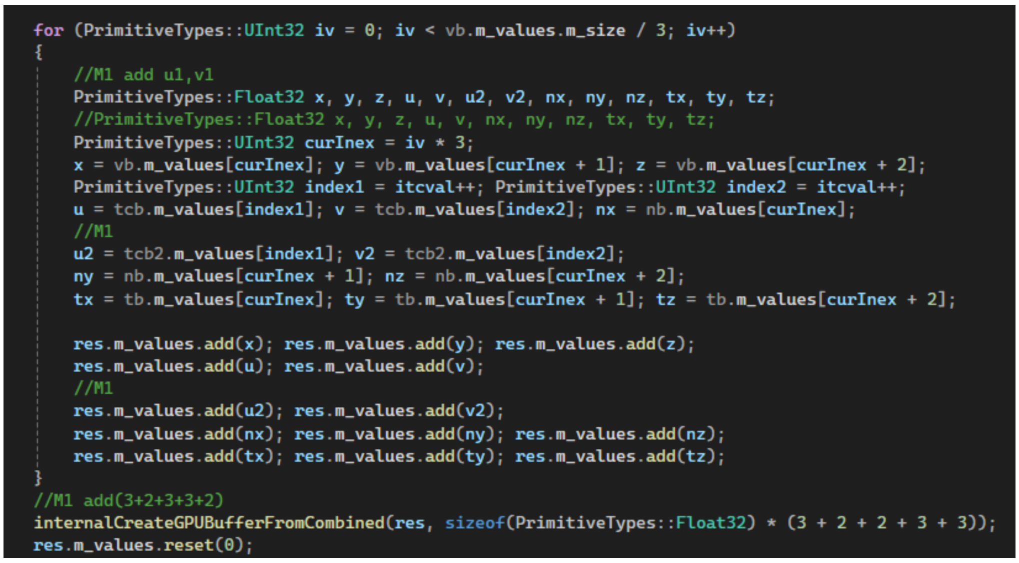
MeshCPU.cpp:



```
//M1
//caltulate y and uv here and pass it to m_hTexCoordBufferCPU1?
    //m_hTexCoordBufferCPU1 = PositionBufferCPUManager::Instance()->ReadTexCoordBuffer(tcfilename, package, tag);
m_hTexCoordBufferCPU1 = Handle("TEXCOORD_BUFFER_CPU", sizeof(TexCoordBufferCPU));
TexCoordBufferCPU* ptcb1 = new(m_hTexCoordBufferCPU1) TexCoordBufferCPU(*m_pContext, m_arena);
Array<PrimitiveTypes::Float32> m_valuesPos = m_hPositionBufferCPU.getObject<PositionBufferCPU>()->m_values;
ptcb1->createMockCPUBuffer(m_valuesPos.m_size / 3);//create tc buf with size of poins *2
float ymin = m_valuesPos[1];
float ymax = m_valuesPos[1];
//get ymax=ymin
for (int i = 0; i < m_valuesPos.m_size; i += 3) {
    if (ymin > m_valuesPos[i + 1])
        ymin = m_valuesPos[i + 1];
    if (ymax < m_valuesPos[i + 1])
        ymax = m_valuesPos[i + 1];
}
//ptcb1->m_values , uv1,uv2,uv,uv...... pass it to v.
for (int i = 0; i < m_valuesPos.m_size / 3; i++) {

    float normalizeY = (m_valuesPos[i * 3 + 1] - ymin) / (ymax - ymin);
    ptcb1->m_values[i * 2] = normalizeY;// here value.x represents the degree been effect by wind
    ptcb1->m_values[i * 2 + 1] = normalizeY;
}
```

This initializes the texture coordinate buffer (m_hTexCoordBufferCPU1) that will be used for the texture mapping of the vertices. It reads position data from a PositionBufferCPU object, which contains the 3D positions of the mesh vertices. Then, it creates a mock CPU buffer for texture coordinates, sized based on the number of vertices. It normalizes the y position of vertices to use as texture coordinates.

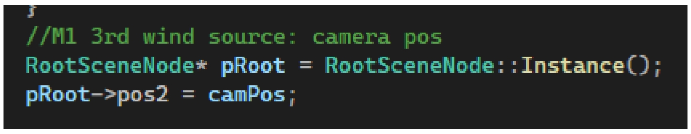
VertexBufferGPU.cpp:



```
for (PrimitiveTypes::UInt32 iv = 0; iv < vb.m_values.m_size / 3; iv++)
{
    //M1 add u1,v1
    PrimitiveTypes::Float32 x, y, z, u, v, u2, v2, nx, ny, nz, tx, ty, tz;
    //PrimitiveTypes::Float32 x, y, z, u, v, nx, ny, nz, tx, ty, tz;
    PrimitiveTypes::UInt32 curInex = iv * 3;
    x = vb.m_values[curInex]; y = vb.m_values[curInex + 1]; z = vb.m_values[curInex + 2];
    PrimitiveTypes::UInt32 index1 = itcval++; PrimitiveTypes::UInt32 index2 = itcval++;
    u = tcb.m_values[index1]; v = tcb.m_values[index2]; nx = nb.m_values[curInex];
    //M1
    u2 = tcb2.m_values[index1]; v2 = tcb2.m_values[index2];
    ny = nb.m_values[curInex + 1]; nz = nb.m_values[curInex + 2];
    tx = tb.m_values[curInex]; ty = tb.m_values[curInex + 1]; tz = tb.m_values[curInex + 2];

    res.m_values.add(x); res.m_values.add(y); res.m_values.add(z);
    res.m_values.add(u); res.m_values.add(v);
    //M1
    res.m_values.add(u2); res.m_values.add(v2);
    res.m_values.add(nx); res.m_values.add(ny); res.m_values.add(nz);
    res.m_values.add(tx); res.m_values.add(ty); res.m_values.add(tz);
}
//M1 add(3+2+3+3+2)
internalCreateGPUBufferFromCombined(res, sizeof(PrimitiveTypes::Float32) * (3 + 2 + 2 + 3 + 3));
res.m_values.reset(0);
```

This prepares the GPU buffer to hold a combination of vertex attributes, including positions, texture coordinates, normals, tangents, etc. It calculates the new size of the buffer, which needs space for positions (3 floats per vertex), texture coordinates (2 floats), normals (3 floats), tangents (3 floats), and other attributes as defined later in the code. Each vertex consists of 3 position components (x, y, z), 2 texture coordinates (u, v), 2 sets of texture coordinates for the wind effect (u2, v2), normal (nx, ny, nz), and tangent (tx, ty, tz). This combination of data is sent to the GPU buffer for later rendering. It creates a GPU buffer by combining vertex positions, texture coordinates, normals, tangents, and other attributes.

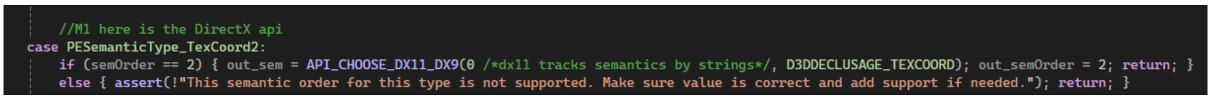
CameraSceneNode.cpp:



```
//M1 3rd wind source: camera pos
RootSceneNode* pRoot = RootSceneNode::Instance();
pRoot->pos2 = camPos;
```

The camera's position is being used as a "wind source" in the scene, likely to influence dynamic effects based on the camera's location.

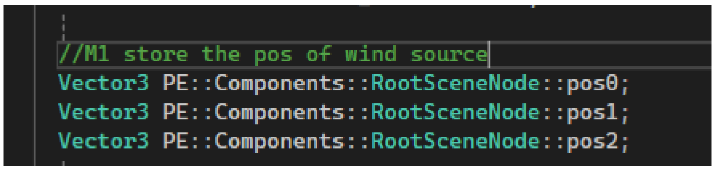
BufferInfo.cpp:



```
    //M1 here is the DirectX api
case PESemanticType_TexCoord2:
    if (semOrder == 2) { out_sem = API_CHOOSE_DX11_DX9(0 /*dx11 tracks semantics by strings*/, D3DDECLUSAGE_TEXCOORD); out_semOrder = 2; return; }
    else { assert(!"This semantic order for this type is not supported. Make sure value is correct and add support if needed."); return; }
```

Code handles the setting of texture coordinate semantics for the second texture coordinate (TexCoord2)

RootSceneNode.cpp:



```
//M1 store the pos of wind source
Vector3 PE::Components::RootSceneNode::pos0;
Vector3 PE::Components::RootSceneNode::pos1;
Vector3 PE::Components::RootSceneNode::pos2;
```

Store the positions of two soldiers and the camera using Vector3 objects. The positions are used to calculate distances and potentially interact with environmental effects like wind

DetailedMesh_Shadowed_A_Glow_PS.cgps:

```
float4 DetailedMesh_Shadowed_A_Glow_PS(DETAILED_MESH_SHADOWED_PS_IN pIn)
{
//M1 create the gradients effect
float3 pos = pIn.iPosW;
 float3 direction0 = pos.xyz - gLight0.xyzPos_w.xyz;
 float3 direction1 = pos.xyz - gLight1.xyzPos_w.xyz;
 float3 direction2 = pos.xyz - gLight2.xyzPos_w.xyz;
    direction0.y = 0;
        direction1.y = 0;
    direction2.y = 0;
    float distance0 = dot(direction0, direction0); // Squared distance
        float distance1 = dot(direction1, direction1);
        float distance2 = dot(direction2, direction2);

return make_float4(pIn.iTexCoord2.y+(1/distance0)+(1/distance1)+(1/distance2),pIn.iTexCoord2.y+(1/distance0)+(1/distance1)+(1/distance2),0, 0);
|

        // note: 0 alpha means no glow
        float4 originalColor = sample2D(gDiffuseMapSampler, pIn.iTexCoord, gDiffuseMap);

        #if APIABSTRACTION_PSVITA
        return originalColor;
        #endif
```

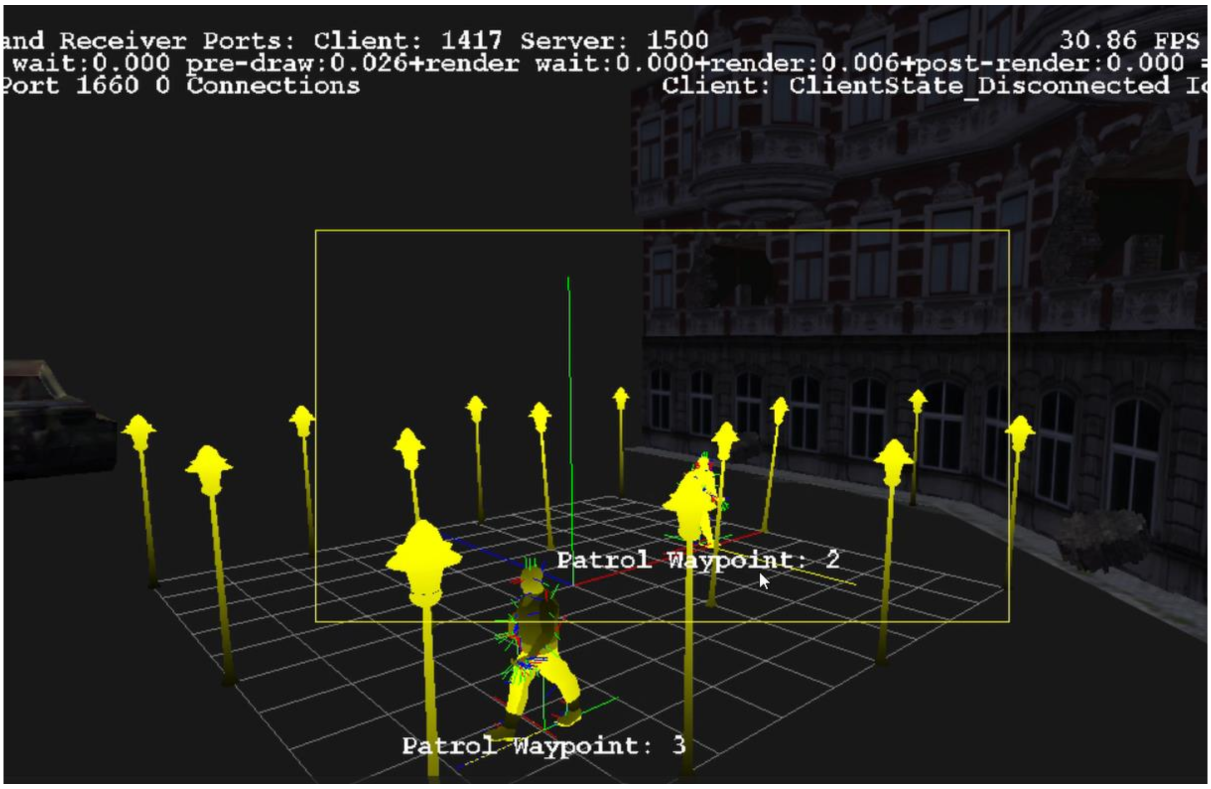
This calculates the direction of the light sources relative to the fragment's position (pos). The y components are set to 0 to ignore vertical variations in the wind effect. The direction vectors from each light source to the fragment are computed, and the squared distances to each light are calculated for later use in light attenuation.

DetailedMesh_Shadowed_VS.cgvs:

```
DETAILED_MESH_SHADOWED_PS_IN DetailedMesh_Shadowed_VS(DETAILED_MESH_VS_IN vIn)
{

    DETAILED_MESH_SHADOWED_PS_IN vOut;
        float3 pos = vIn.iPosL;

        Matrix WVP = gWVP;
        Matrix W = gW;

    //M1
    // Calculate the direction vector from the current position to wind position
    float3 direction0 =  mul(make_float4(pos, 1.0), W).xyz - gLight0.xyzPos_w.xyz;
    float3 direction1 =  mul(make_float4(pos, 1.0), W).xyz - gLight1.xyzPos_w.xyz;
    float3 direction2 =  mul(make_float4(pos, 1.0), W).xyz - gLight2.xyzPos_w.xyz;
    direction0.y = 0;
    direction1.y = 0;
    direction2.y = 0;

    float distance0 = dot(direction0, direction0); // Squared distance
    float distance1 = dot(direction1, direction1);
    float distance2 = dot(direction2, direction2);
    direction0 = normalize(direction0); // Normalize the direction vector
    direction1 = normalize(direction1);
    direction2 = normalize(direction2);

    float magnitude = vIn.iTexCoord2.x * 1.2; // The wind influence factor from the second texture coordinate

    // Apply the wind effect based on the distance and magnitude
    pos = pos + (1/distance0) * magnitude  * direction0;
    pos = pos + (1/distance1) * magnitude  * direction1;
    pos = pos + (1/distance2) * magnitude  * direction2;




    vOut.iPosH = mul(make_float4(pos, 1.0), WVP);
    vOut.iNormalW = mul(make_float4(vIn.iNormal, 0), W).xyz;
    vOut.iPosW =  mul(make_float4(pos, 1.0), W).xyz;
    vOut.iTangentW = mul(make_float4(vIn.iTangent, 1.0), W).xyz;
    vOut.iTexCoord = vIn.iTexCoord;

    //M1 add another uv map, u=0, v=y
    //vOut.iTexCoord2 = float2(0, 0.3);
    vOut.iTexCoord2 = vIn.iTexCoord2;

    vOut.iProjTexCoord = mul(make_float4(vOut.iPosW, 1.0), gLightWVP);
    return vOut;
}
```

This returns the final color for the pixel based on the texture coordinates and distance-based attenuation for each light source. The y texture coordinate (pIn.iTexCoord2.y) is modified by the inverse of the distances to the light sources to simulate the effect of the wind on the color. The resulting color is used for the fragment's final appearance, which may be affected by the lighting and wind influence. The fragment shader applies lighting from multiple sources and wind effects based on distance and texture coordinates, affecting the final color of the mesh.

Output



```
and Receiver Ports: Client: 1417 Server: 1500                    30.86 FPS
 wait:0.000 pre-draw:0.026+render wait:0.000+render:0.006+post-render:0.000 =
Port 1660 0 Connections                         Client: ClientState_Disconnected Id
```