CSCI 522 – Game Engine - Fall 2024 - HW 3

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

1) Creating bounding volumes:

Take the m_values in the PositionbufferCPU and find the max xyz and min xyz values for the AABB vertices calculation.

```
Vector3 firstVertex(m_value[0], m_value[1], m_value[2]);
m_AABB.min = firstVertex;
m_AABB.max = firstVertex;

for (int i = 1; i < m_value.m_size / 3; i++) {
    Vector3 vertex(m_value[i * 3], m_value[i * 3 + 1], m_value[i * 3 + 2]);

    m_AABB.min.m_x = std::min(m_AABB.min.m_x, vertex.m_x);
    m_AABB.min.m_y = std::min(m_AABB.min.m_y, vertex.m_y);
    m_AABB.min.m_z = std::min(m_AABB.min.m_z, vertex.m_z);

    m_AABB.max.m_x = std::max(m_AABB.max.m_x, vertex.m_x);
    m_AABB.max.m_y = std::max(m_AABB.max.m_y, vertex.m_y);
    m_AABB.max.m_z = std::max(m_AABB.max.m_z, vertex.m_z);
}
```

Mesh::calculateAABB() use this function to calculate AABB min and max

```
void updateCorners() {
    corners[0] = min;
    corners[1] = Vector3(min.m_x, min.m_y, max.m_z);
    corners[2] = Vector3(min.m_x, max.m_y, min.m_z);
    corners[3] = Vector3(min.m_x, max.m_y, max.m_z);
    corners[4] = Vector3(max.m_x, min.m_y, min.m_z);
    corners[5] = Vector3(max.m_x, min.m_y, max.m_z);
    corners[6] = Vector3(max.m_x, max.m_y, min.m_z);
    corners[7] = max;
}
```

Struct AABB::updateCorners takes min and max and find the vertices for the boundinbox

2) Showing bounding volumes:

```
const int numofLines = 12;
const int numofVerticesPerLine = 2;
Vector3 lines[numofLines * numofVerticesPerLine * 2];
int segments[numofLines][2] = {
    {0, 1}, {1, 3}, {3, 2}, {2, 0},
    {4, 5}, {5, 7}, {7, 6}, {6, 4},
    {0, 4}, {1, 5}, {2, 6}, {3, 7}
};
Vector3 defaultColor(1.0f, 1.0f, 0.0f);
int idx = 0;
for (int i = 0; i < numofLines; i++) {
    for (int j = 0; j < numofVerticesPerLine; j++) {
        if (isCornerInicialized) {
            lines[idx++] = corners[segments[i][j]];
            lines[idx++] = defaultColor;
        }
    }
}
createLineMesh(false, Matrix4x4(), &lines[0].m_x, numofLines * numofVerticesPerLine, timeToLive);
```

DebugRenderer::debugRenderBox to render lines for the boundingbox

```
PE::Components::AABB AABB_Box = pMeshCaller->m_AABB;
Handle parentSceneNode = pInst->getFirstParentByType<SceneNode>();
Matrix4x4 base = parentSceneNode.getObject<SceneNode>()->m_worldTransform;
PE::Components::AABB AABB(base * AABB_Box.max, base * AABB_Box.min);
pInst->m_transformedAABB = AABB;
debugRenderBox(AABB.corners, timeToLive, isCornerInicialized);
```

DebugRenderer::debugRenderAABB it takes mesh and transforms it based on the parent scene node's world transform

3) Creating Frustum Plane Equations:

Components::Struct plane

Components::Struct Frustum

```
PrimitiveTypes::Float32 tanHalfFov = tanf(verticalFov / 4.0f);
PrimitiveTypes::Float32 halfHeightNear = m_near * tanHalfFov;
PrimitiveTypes::Float32 halfWidthNear = halfHeightNear * aspect;
PrimitiveTypes::Float32 halfHeightFar = m_far * tanHalfFov;
PrimitiveTypes::Float32 halfWidthFar = halfHeightFar * aspect;

Vector3 camPos = Vector3(m_worldTransform.m[0][3], m_worldTransform.m[1][3], m_worldTransform.m[2][3]);
Vector3 camDir = Vector3(m_worldTransform.m[0][2], m_worldTransform.m[1][2], m_worldTransform.m[2][2]);
Vector3 camUp = Vector3(m_worldTransform.m[0][1], m_worldTransform.m[1][1], m_worldTransform.m[2][1]);
Vector3 camRight = camDir.crossProduct(camUp);

Vector3 nearCenter = camPos + camDir * m_near;
Vector3 farCenter = camPos + camDir * m_far;

frustum.corners[0] = nearCenter + halfHeightNear * camUp - halfWidthNear * camRight;
frustum.corners[1] = nearCenter + halfHeightNear * camUp + halfWidthNear * camRight;
frustum.corners[2] = nearCenter - halfHeightNear * camUp - halfWidthNear * camRight;
frustum.corners[3] = nearCenter - halfHeightNear * camUp + halfWidthNear * camRight;
frustum.corners[4] = farCenter + halfHeightFar * camUp - halfWidthFar * camRight;
frustum.corners[5] = farCenter + halfHeightFar * camUp + halfWidthFar * camRight;
frustum.corners[6] = farCenter - halfHeightFar * camUp - halfWidthFar * camRight;
frustum.corners[7] = farCenter - halfHeightFar * camUp + halfWidthFar * camRight;

frustum.planes[0] = Plane(frustum.corners[3], frustum.corners[2], frustum.corners[0]);
frustum.planes[1] = Plane(frustum.corners[4], frustum.corners[6], frustum.corners[7]);
frustum.planes[2] = Plane(frustum.corners[6], frustum.corners[2], frustum.corners[3]);
frustum.planes[3] = Plane(frustum.corners[1], frustum.corners[0], frustum.corners[4]);
frustum.planes[4] = Plane(frustum.corners[0], frustum.corners[2], frustum.corners[6]);
frustum.planes[5] = Plane(frustum.corners[7], frustum.corners[3], frustum.corners[1]);
```

CameraSceneNode::frustumValues It computes the corners and planes based on camera properties

4) Testing bounding volumes:

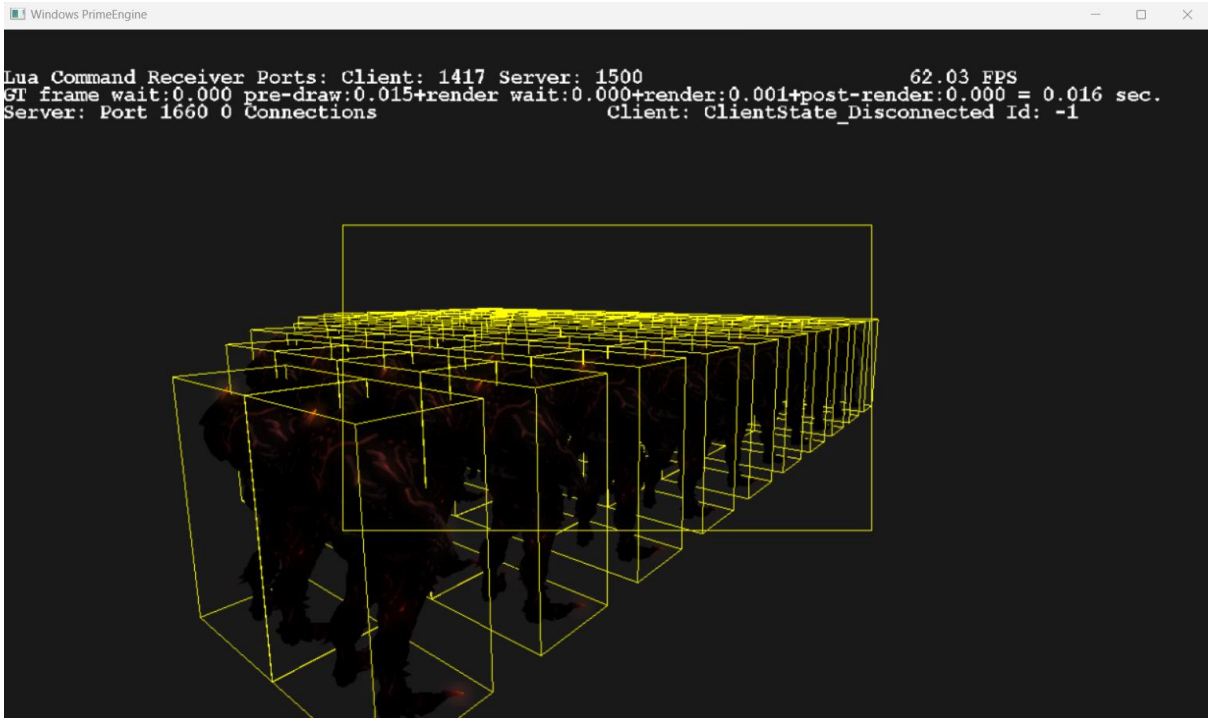
```
bool inside = true;
if (pDrawEvent != NULL && pDrawEvent->isCornerInitialized) {

    AABB AABB = pInst->m_transformedAABB;
    float* distances = pDrawEvent->planeDistances;
    Vector3* normals = pDrawEvent->planeNormals;

    for (int i = 0; i < 6; ++i) {
        int outcount = 0;
        for (int j = 0; j < 8; ++j) {
            if (normals[i].dotProduct(AABB.corners[j]) + distances[i] < 0.0f) {
                outcount++;
            }
        }
        if (outcount == 8) {
            inside = false;
        }
    }
    if (inside)
    {
        pInst->m_culledOut = false;
        ++pMeshCaller->m_numVisibleInstances;
        DebugRenderer::Instance()->debugRenderAABB(pInst, pMeshCaller, 0.2f, true);
    }
    else {
        pInst->m_culledOut = true;
    }
}
```

SingleHandler_DRAW::do_GATHER_DRAWCALLS It uses the dot product to determine the position of each corner of the AABB relative to the planes. If the AABB is entirely within the frustum, it is marked as visible; if not, it is culled from rendering.

5) Testing your code:


Windows PrimeEngine
Lua Command Receiver Ports: Client: 1417 Server: 1500
62.03 FPS
GT frame wait:0.000 pre-draw:0.015+render wait:0.000+render:0.001+post-render:0.000 = 0.016 sec.
Server: Port 1660 0 Connections
Client: ClientState_Disconnected Id: -1