Duck Hunt CAVE

Christoph Winter*

Technische Universitt Mnchen

Abstract

This project was done for the Virtual Reality course of LMU in 2017. The goal was to create an virtual reality application that runs on the LRZ CAVE, has to be interactive and must run in real-time. The program was written in C++ with the use of OpenSG, OpenGL with GLUT, inVRs and VRPN. With these tools and goal in mind, I created a clone of the all-time classic game Duck Hunt.

Index Terms: Virtual Reality—CAVE—OpenSG—Duck Hunt

1 Topic Overview

Duck Hunt is a classic Nintendo game from 1984. The game mechanics are quite simple: You have a limited number of shots and need to shoot the birds, but not the dog. Each killed bird increases your score, when you are out of ammunition, the game is over. I uses this concept to make my own Duck Hunt clone which should work in the LRZ CAVE.

2 User Guide

The wand is used to aim and shoot the ducks. You pull the trigger of the gun with the button on the back of the wand. The crosshair is a little help to display your target position.

3 Implementation Concept

3.1 Creating the Engine of the Game

The most important part of a game engine is to be able to receive the time between two frames to be able to move objects relative to the time. Thus, I created a class called MyTime, which holds relevant timing information, such as the scaled and unscaled delta time between two frames, the amount of frames rendered so far, and the current time scale (for slowing/accelerating time).

```
high_resolution_clock::time_point MyTime::
    lastUpdateTimepoint = high_resolution_clock::
    now();

void MyTime::UpdateDeltaTime()
{
    double deltaTime = duration_cast<duration<
        double>>(high_resolution_clock::now() -
        lastUpdateTimepoint).count();
    DeltaTime = deltaTime * TimeScale;
    UnscaledDeltaTime = deltaTime;
    RealtimeSinceStartup += deltaTime;
    FrameCount++;
    lastUpdateTimepoint = high_resolution_clock::
        now();
}
```

Furthermore, a class that can receive an update per frame was needed. Additionally, it should generate a ComponentTransform with a Group as a child, so it can be positioned in the world and can any amount of have children. This class is called GameObject. The Game keeps track of all instances of a GameObject and calls the Update function on them.

```
void Game::Update()
{
    ...
    // Update all behaviors
    for (auto& behavior : m_behaviors)
    {
        behavior.second->Update();
    }
    ...
}
```

The glutDisplayFunc calls the Update function of the game, so that each frame the game and all gameobjects in it are updated.

```
glutDisplayFunc([]()
{
    MyTime::UpdateDeltaTime();
    ...
    game->Update();
    // black navigation window
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers();
});
```

Each GameObject can have Components attached to it, which are mostly Geometry nodes. Currently, two types of components are supported: Mesh and Sprite. A Mesh loads a 3D Object from the disk and creates its geometry. This is currently only used for the Gun. A Sprite creates a plane geometry and sets the image as the texture of that Sprite for each sprite in the specified sprite list. All images are added to a Switch node, so that one image can be activated at a time without having to create a new geometry.

*e-mail: c.winter@tum.de

```
Sprite::Sprite(...)
{
...
// create one geometry + texture per sprite in the
     sprite list, and key this information for
    later use
int spriteSwitcherIndex = 0;
for (auto& sprite : m_SpriteList)
{
   for (size_t i = 0; i < sprite.second.size(); i
       ++)
   {
      const std::string spriteName = sprite.second
          [i];
      m_SpriteIdToIndexMap[spriteName] =
          spriteSwitcherIndex++;
      const ImageRecPtr image = m_SpriteAtlas->
          GetImage(spriteName);
      SimpleTexturedMaterialRecPtr material =
          SimpleTexturedMaterial::create();
      material->setImage(image);
      material->setSortKey(sortKey);
      GeometryRecPtr spriteGeo = makePlaneGeo(
          image->getWidth()+1, image->getHeight()
          +1, 1, 1);
      NodeRecPtr spriteGeoNode = Node::create();
      spriteGeoNode->setCore(spriteGeo);
      spriteGeo->setMaterial(material);
      m_SpriteSwitcher.node()->addChild(
          spriteGeoNode);
   }
}
}
```

Another important component is the SpriteAtlas. This class is responsible for reading a sprite sheet and creating an Image for each sprite in the sheet. It also generates a list for each sprite animation, which the Sprite class can read and display.

3.2 Implementing the Game Mechanics

Now, it is possible to display Sprites and move them in the scene. The next step was to make the birds fly and be able to shoot them.

The position of each duck gets updated every frame depending on the delta time, speed and direction it is flying in. The position of the bird is kept inside a specific range so that you can always shoot it.

Shooting a duck works by shooting a ray from the barrel exit into the direction the gun is looking. If the ray intersects with a duck geometry, the Game checks which Bird was hit and calls the OnHit() function of that duck. The duck executes his hit and dead animation and gets removed from the scenegraph. This continues until all ducks are dead, then new ones are spawned.

3.3 Future Work

The clone still has some missing features from the original Duck Hunt game. There is no score, there are no dogs and you can't lose. Although all assets would be available, they are not implemented yet due to lack of time.

Also, the floor should have been sand using a wrapping texture, but I got not manage to get the texture showing, so it stayed black.

4 Screenshots

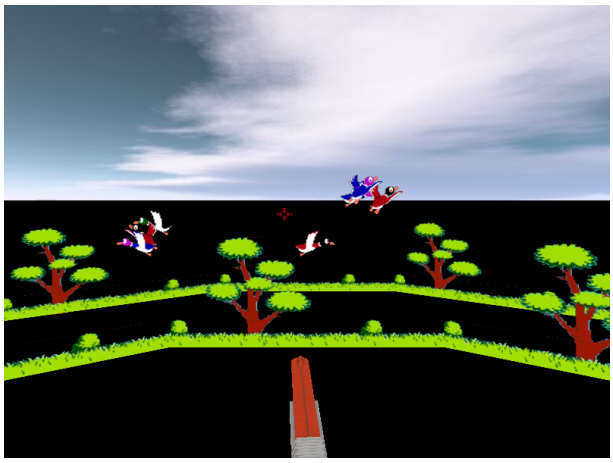

Figure 1: Shoot the ducks!

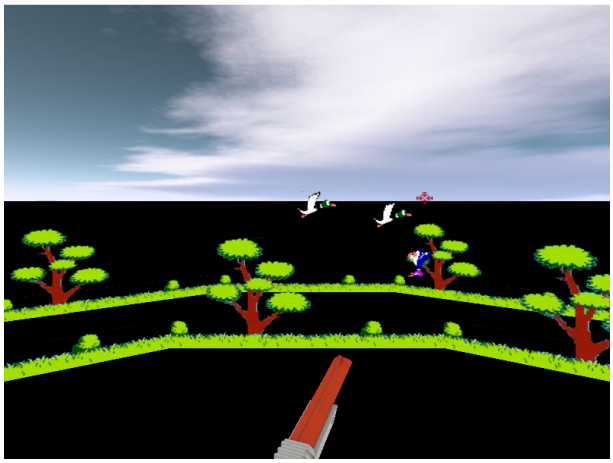

Figure 2: A duck was hit!