California Polytechnic State University, San Luis Obispo

# Real-Time Volumetric Cloud Rendering

Jaafer Sheriff

Senior Project

June 2018

Supervisor
Dr. Christian Eckhardt

# ABSTRACT

short abstract

## Contents

## 1 Introduction

introduction/motivation (1 page)

# 2 Implementation

We considered 3 major steps to create our clouds. The first was spherical voxelization of billboards, the second was voxel cone tracing, and the third was noise generation.

## 2.1 Voxelization

Normally a volume is populated using some provided static 3D texture data. For our implementation we manually update every voxel ourselves.

The goal of voxelization is to properly populate our volume. We want our voxels to have a billowing, semi-spherical, cloud-like shape. Secondly, we want to highlight the voxels nearest to the light source.

We do our voxelization using billboards and a multi-pass voxelization system. We also clear the entire volume and revoxelize on every frame. This will be essential if we ever add procedurally animated clouds.

### 2.1.1 Spherical Billboards

We begin by rendering many light-facing billboards. In the billboard's fragment shader we calculate a spherical distribution mapped to the face of the billboard. Using this spherical distribution, we populate a sphere in the volume.

```
// Calculate distance from current fragment to billboard's center
float dist = distance(center, fragPos);
// Calculate linear distribution
float distribution = distToCenter / radius;
// Convert linear distribution to spherical
distribution = sqrt(max(0, 1 - distribution * distribution));
```

Listing 1: first_voxelize.glsl, 42

Figure 1: Spherical distribution on a billboard

### 2.1.2 Position Map

When rendering these light-facing billboards, we simulatenously write the positions of the sphere into a frame buffer. The resulting frame buffer gives us the world positions of the voxels nearest to the light source. Initially we expected the depth test to manage overlapping spheres so we would be provided with an accurate position map. However, the depth test uses rasterized geometry, and the geometry being rendered is the float billboard, not the spherical representations that we want.

To solve this, we calculate the fragment's spherical depth and manually update the depth buffer. We do this in world-space by using the light positions, the fragment's world position, and the size of the light's orthographic frustum. It would likely be more elegant to do this calculation in clip-space.

```glsl
// Calculate sphere fragment's world position
// This is the position on the sphere nearest to the light source
vec3 worldPos = fragPos + billboardNormal * distribution;
// Calculate sphere fragment's depth
float depth = distance(lightNearPlane, worldPos) /
    distance(lightFarPlane, lightNearPlane);
// Write to frame buffer
outColor = vec4(worldPos, 1);
gl_FragDepth = depth;
```

Listing 2: first_voxelize.glsl, 60

### 2.1.3 Highlighting voxels

The last step for our voxelization technique is to higlight voxels nearest to the camera. Lucky for us, we have exactly what we need in the position map! For our second voxelization pass we render a full screen quad containing the position map. We sample each fragment of the position map to get the voxel position of the voxels nearest to the light source. We then update the voxel data to highlight it.

Our result is a billowing spherical cloud-like structure with voxels nearest to the camera being highlighted.

### 2.1.4 Optimizations

Because of this, our voxels only ever have three states: inactive, active, or highlighted. To optimize our approach, our voxels use R8UI as the internal format. This bits of the red channel are flipped as necessary allowing us to have minimal data representing our volume.

## 2.2 Voxel Cone Tracing

implementation (5-6 pages) where you argue in detail how you implemented what i.e. code

## 2.3 Noise Generation

### 2.3.1 Parameter playing

implementation (5-6 pages) where you argue in detail how you implemented what i.e. code

## 3 Results

3 page result (including pictures).