

# Adaptive Point Cloud Rendering

May13-11

**Members:**  
Joel Rausch CprE  
Eric Jensen SE/ME  
Christopher Jeffers CprE  
**Faculty Advisor:** Dr. Simanta Mitra

**Client:**  
Michael Carter  
Andreas Johannsen  
Siemens PLM Software

## Abstract:

Digital 3D models are used in industry during the design process. Our client, Siemens PLM, creates software to allow these businesses to view and compare 3D models. One type of 3D model a business may obtain is called a point cloud, which is a large group of points in (x,y,z) space with no connections between the points. Most hardware and software is optimized to handle the far more common polygon-based 3D models, and Siemens PLM currently has no program capable of rendering large models made completely of points.

Our task was to design and create a program to render (draw on screen) very large point clouds (hundreds of millions to a billion points). Our design involves a process of converting the point data into a usable structure, going through this structure efficiently to find which points should be drawn, and then efficiently representing the points on the screen.

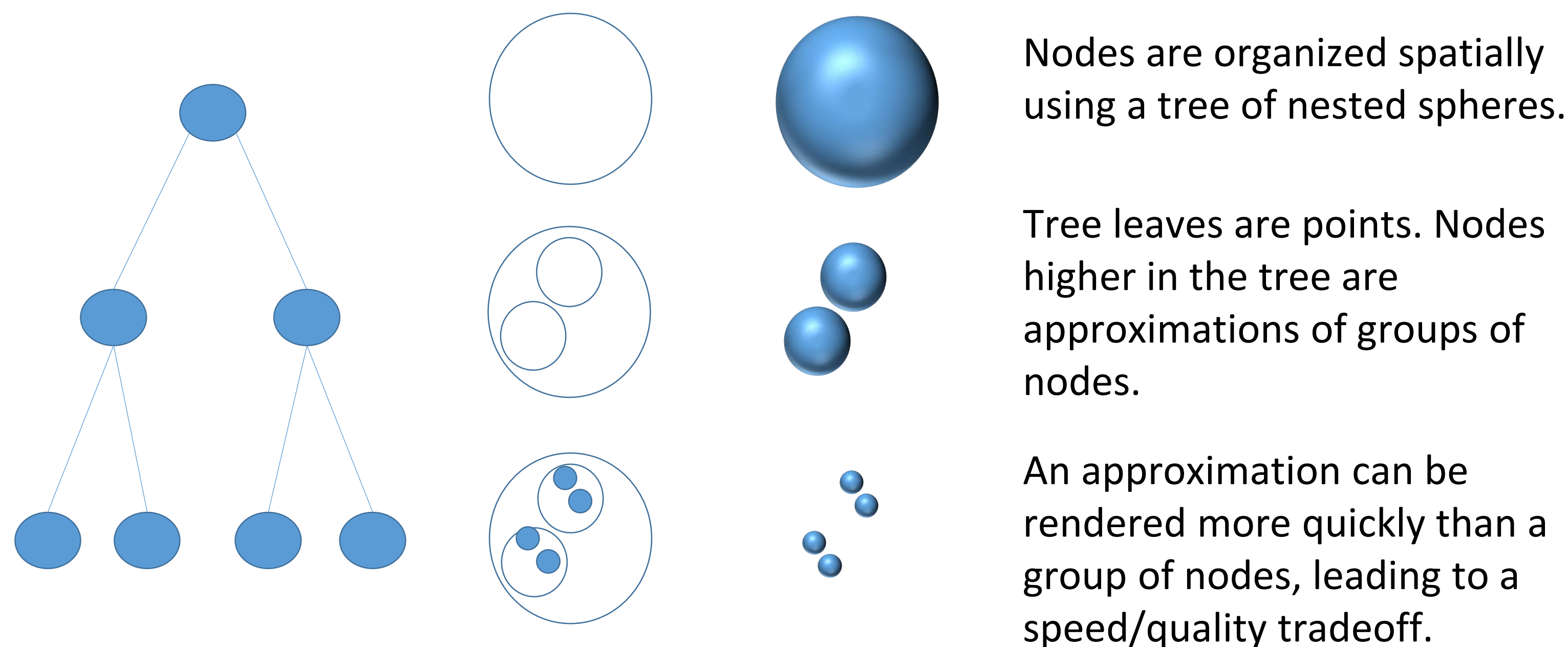
## Problem Statement:

Rendering very large point clouds comes with the challenges of handling memory and making sure we maintain a frame rate high enough to allow the user to interact with the model. We need a solution that will not use a huge amount of system RAM and can quickly decide which points need to be rendered.

## Solution:

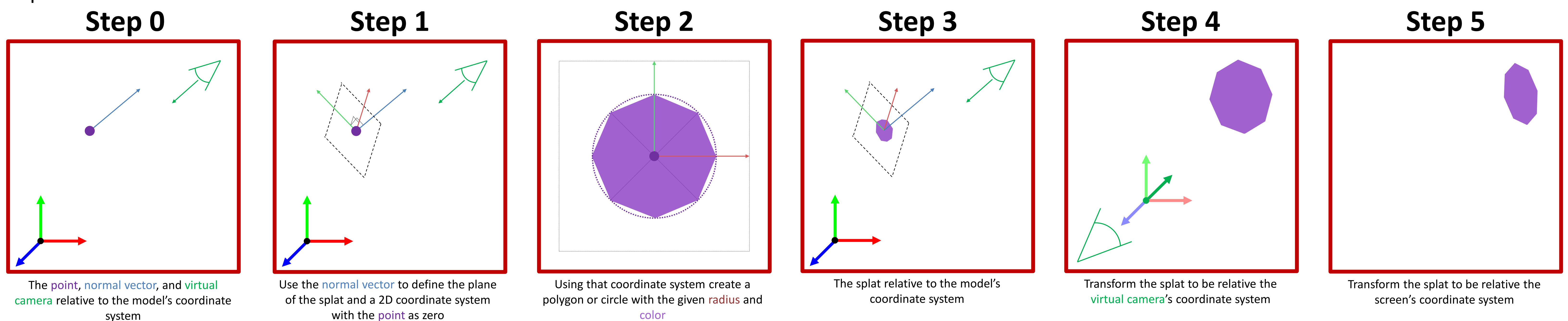
Our research lead us to a point cloud rendering system created at Stanford University in the year 2000, called QSplat. This system creates a very efficient and compressed tree structure for our point data that allows us to keep system memory usage low. Traversing this tree structure allows us to easily find where a point should be rendered. We are able to improve upon their methods because of improved hardware they did not have while creating QSplat. This allowed us to reach our goal of rendering a point cloud with a billion points.

## Level of Detail Progression



## Rendering:

As world state, we have a **virtual camera** defined by three orthonormal vectors and a position. From the traversal, we get a set of points, each with a **position**, a **radius**, a **normal vector** representing the surface, and a **color**. For each point, we create a representation for its surface, a splat, by following steps 1-5. This is done for all points once per frame.



## Visual Results:

From left-to-right, our baseline result, our balanced result, and our highest quality result using the Stanford lion (approx. 180,000 points).



OpenGL Points (Size: Right-1, Left-3)



Circles (Billboard)



Hexagons Blended

## Project Requirements:

### Functional Requirements:

- Load a billion-point point cloud provided by Siemens
- Allow the user to rotate, zoom, and pan on a point cloud model

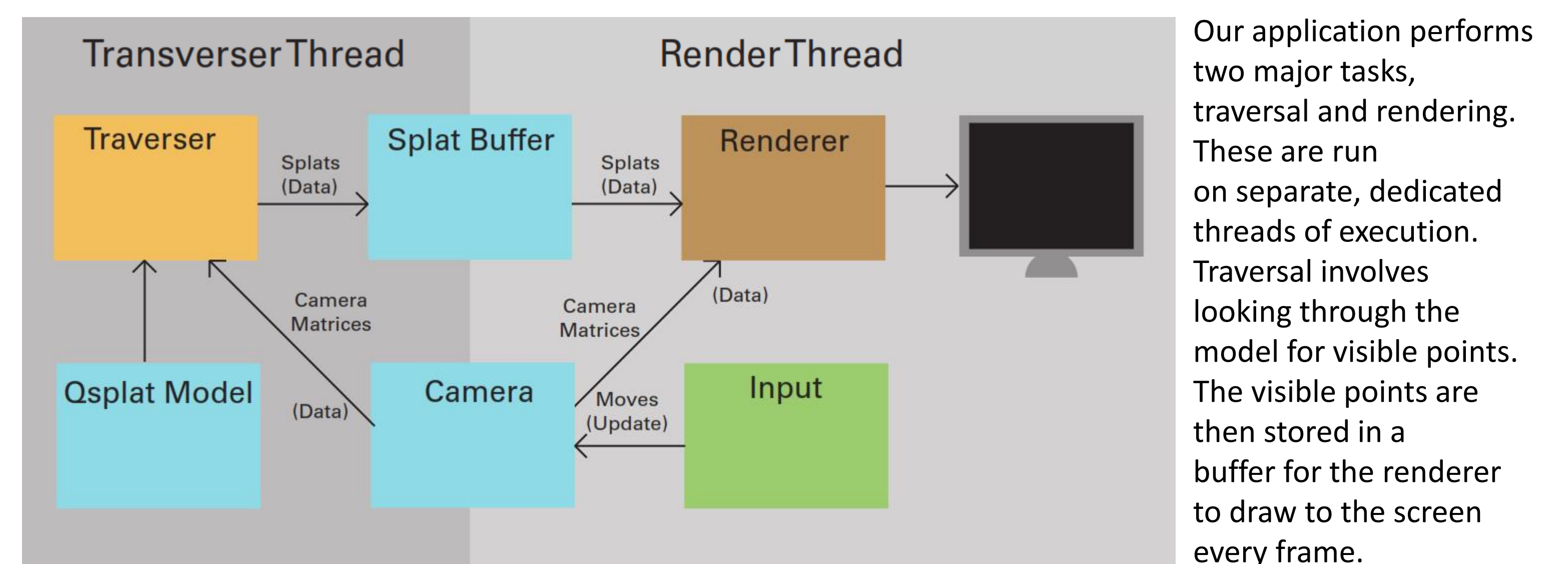
### Non-Functional Requirements:

- Maintain an interactive frame rate of no less than 10 fps at 1080p
- Render the highest quality representation of the point cloud
- Programmed using C++

### Operating Environment:

- Windows PC with graphics card supporting OpenGL 3.3 or higher

## Block Diagram:



## Results:

Using our QSplat variant, we successfully decoupled the size of the point cloud from the frame rate. We developed six types of splats to balance between quality and rendering time. Blending splats together produces the highest quality, but the slowest rendering time. Our circle splat produced the highest quality and was the fastest of our non-blending types.

The chart below shows the frames rates produced for each type of splat when rendering a billion-point cloud at a resolution of 1920x1080.

