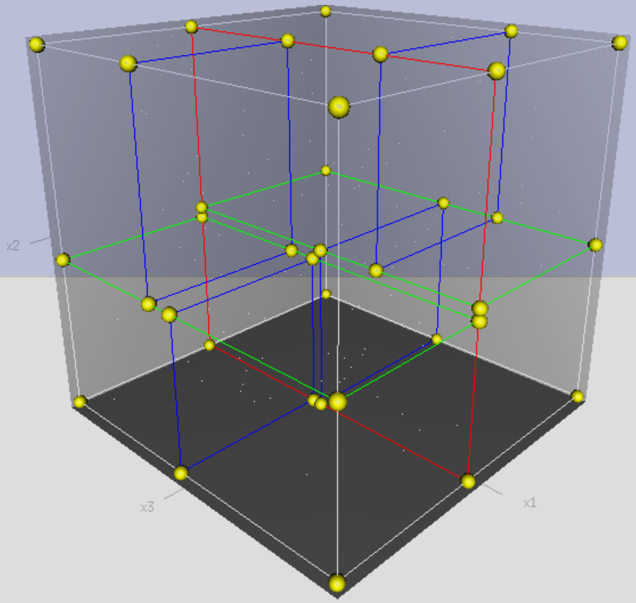
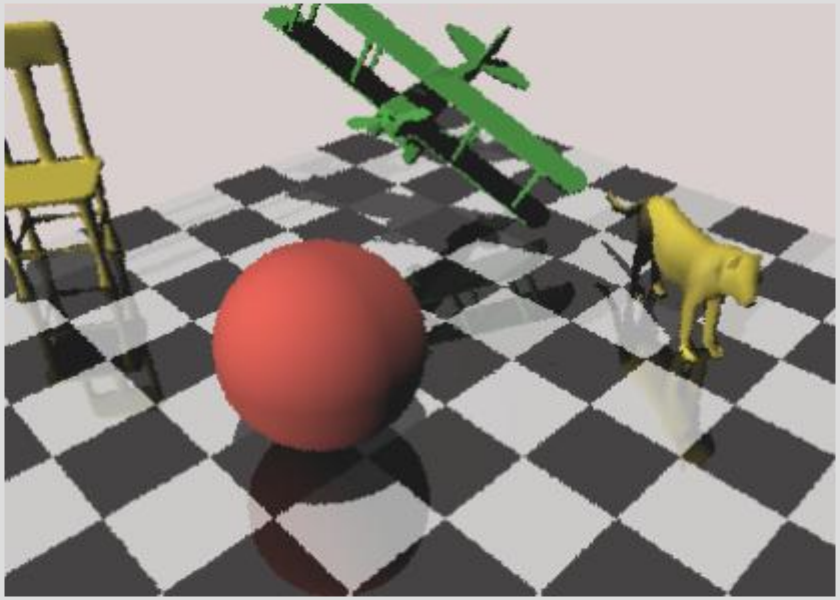
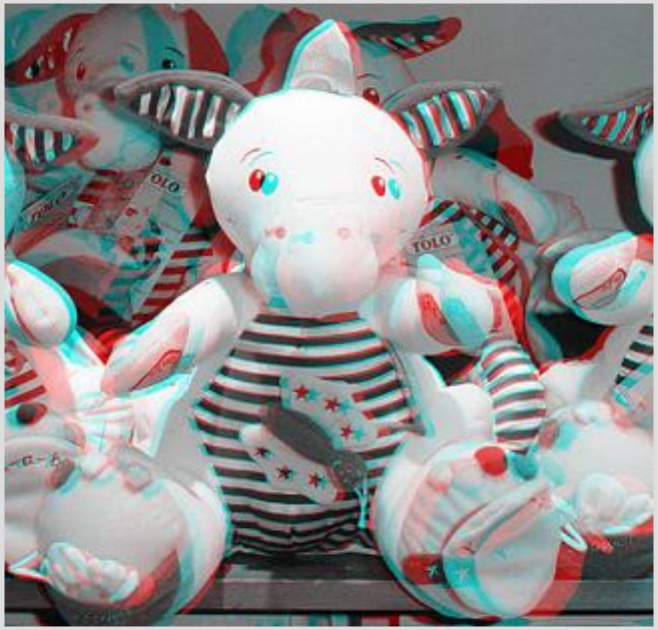
3D Ray Tracing and Image Generation







Thomas Baron | Joshua Bell
John Davis | Nathaniel Williams

Objectives:

- Render complex 3D anaglyph scenes, viewable using 3D glasses
- Create support necessary for such complex scenes
  - Mesh Support
  - Acceleration structures

Anaglyph Images

Rendering Anaglyph Images

Why do it?

What is it?

How is it done?

Our progress

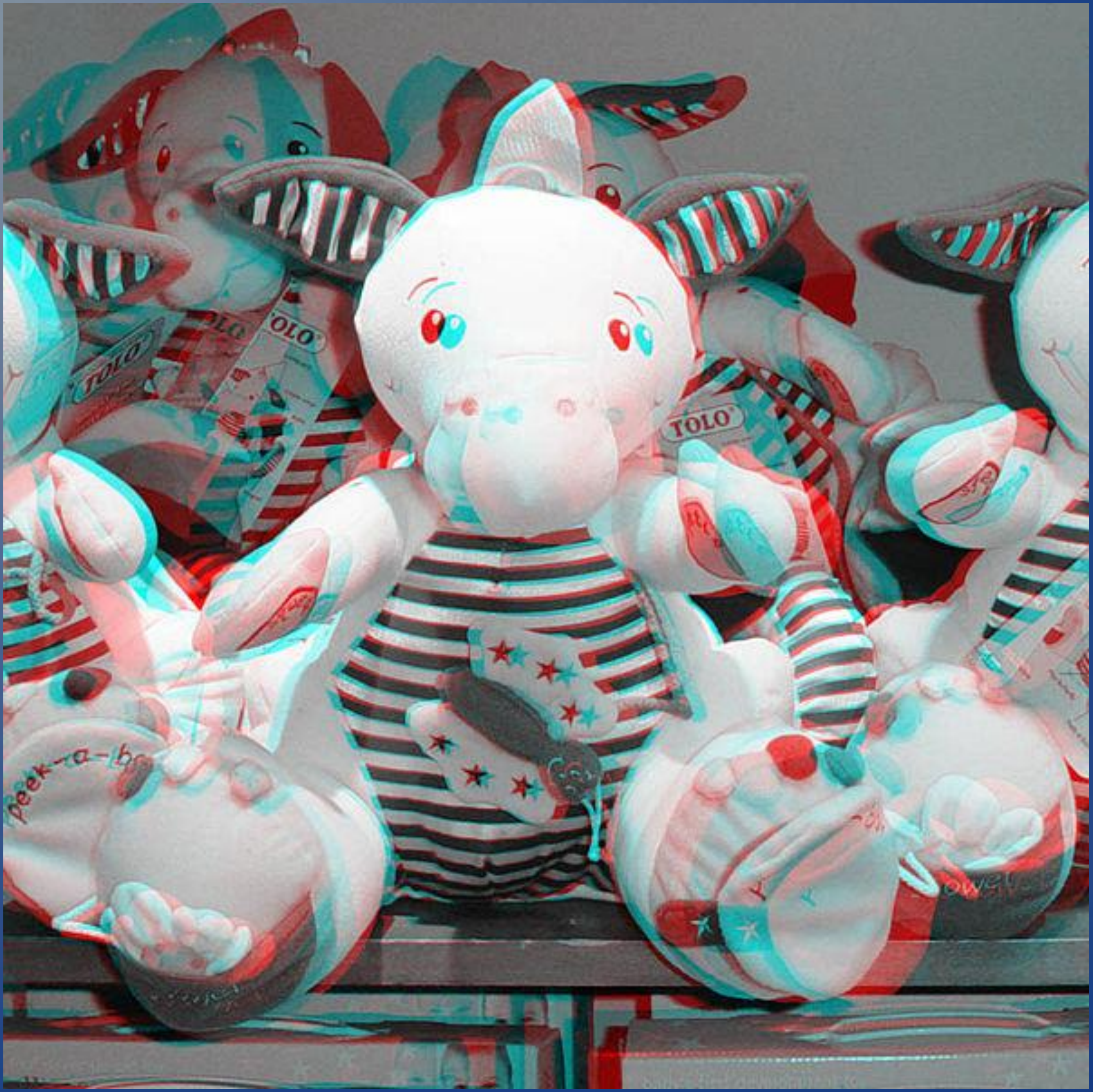
Measuring Success

Resources

Why Anaglyph

- Turn a 2D screen into a 3D display
- Can be implemented with ray tracing effectively
  - Some methods less accurate
- Interesting technical challenge

What is Anaglyph



What is Anaglyph

- Definition
  - A redish left image and a blueish (cyan) right image superimposed onto each other and when viewed with glasses of corresponding colors appears 3D
- Why red and cyan

How to Anaglyph

- Lots of methods available
- Generalized Process
  - Fully ray trace a center and right image
  - Apply post processing to the images

Ray Tracing Both Images

Info: eye dominance

Eye Positions

offsets

View frames

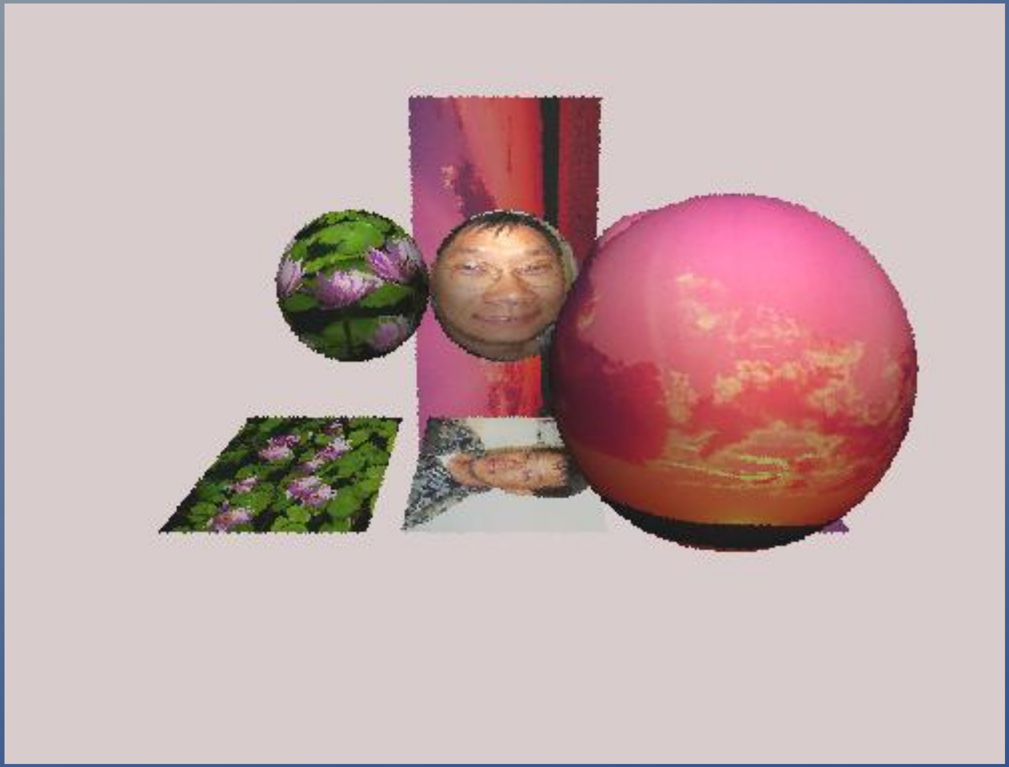
Vanishing point

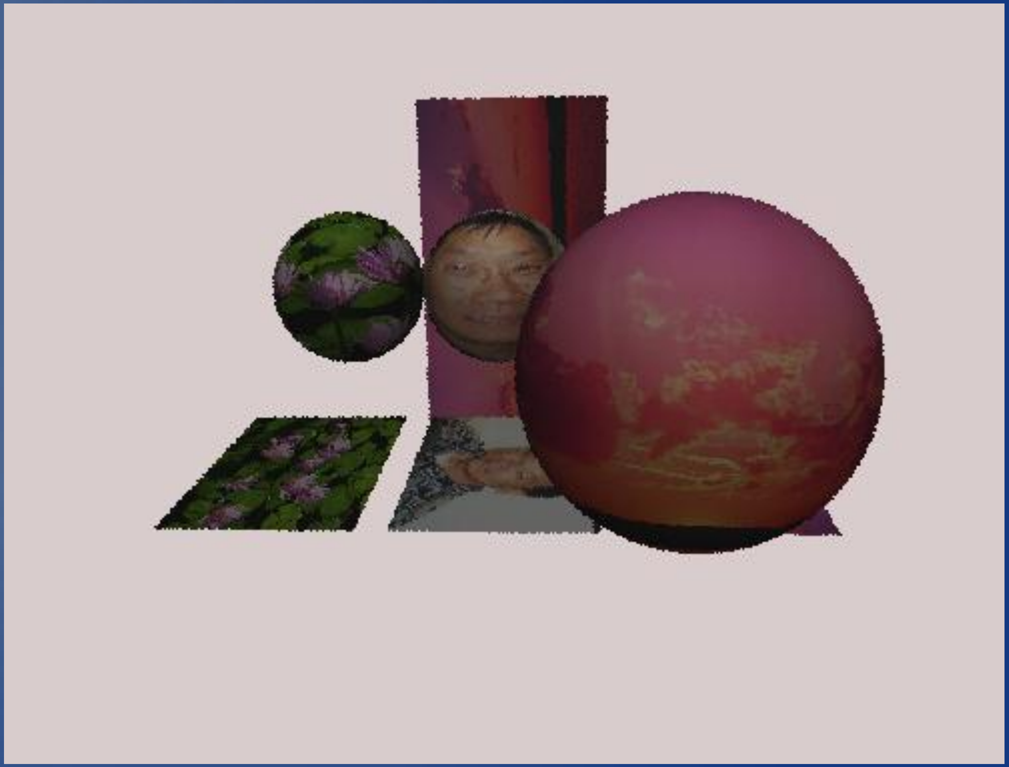
Background objects vs foreground objects

Post Ray Tracing Processing

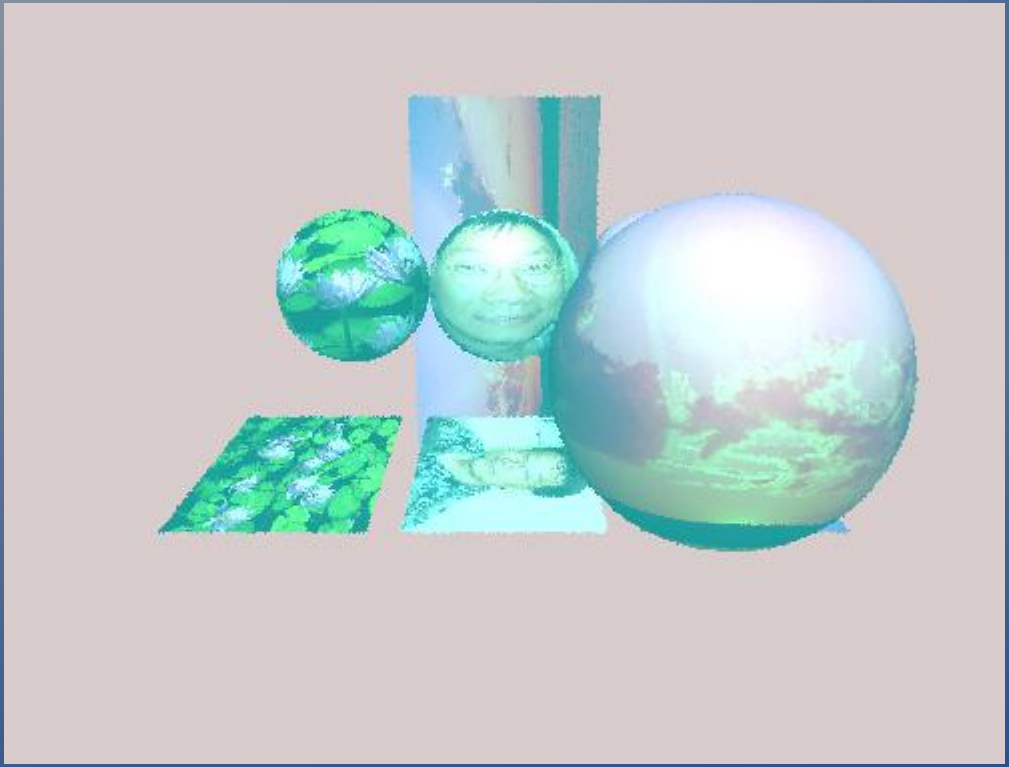
- From Color to red or cyan
  - Use gray scale as a step between full color and red or cyan
  - From practice we have found emphasizing the red and emphasizing the blue in the corresponding images works better
  - Part art. Various methods. Including the retention of green.
- Add corresponding pixel colors together

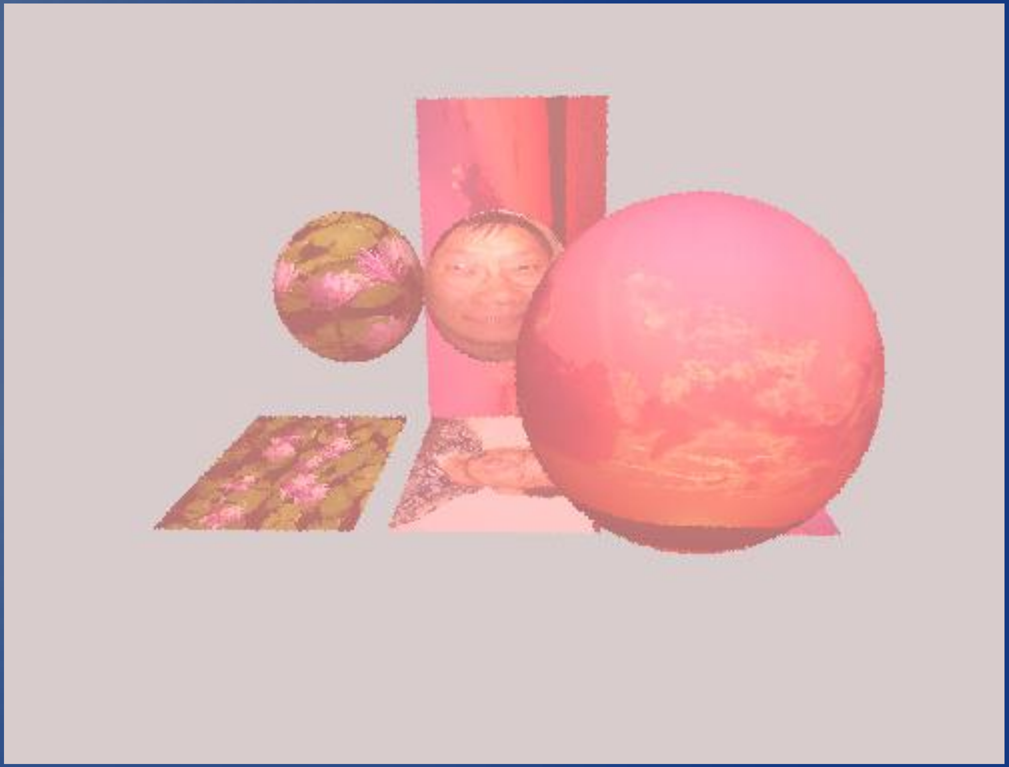
Our Progress: 2 images


Original Center

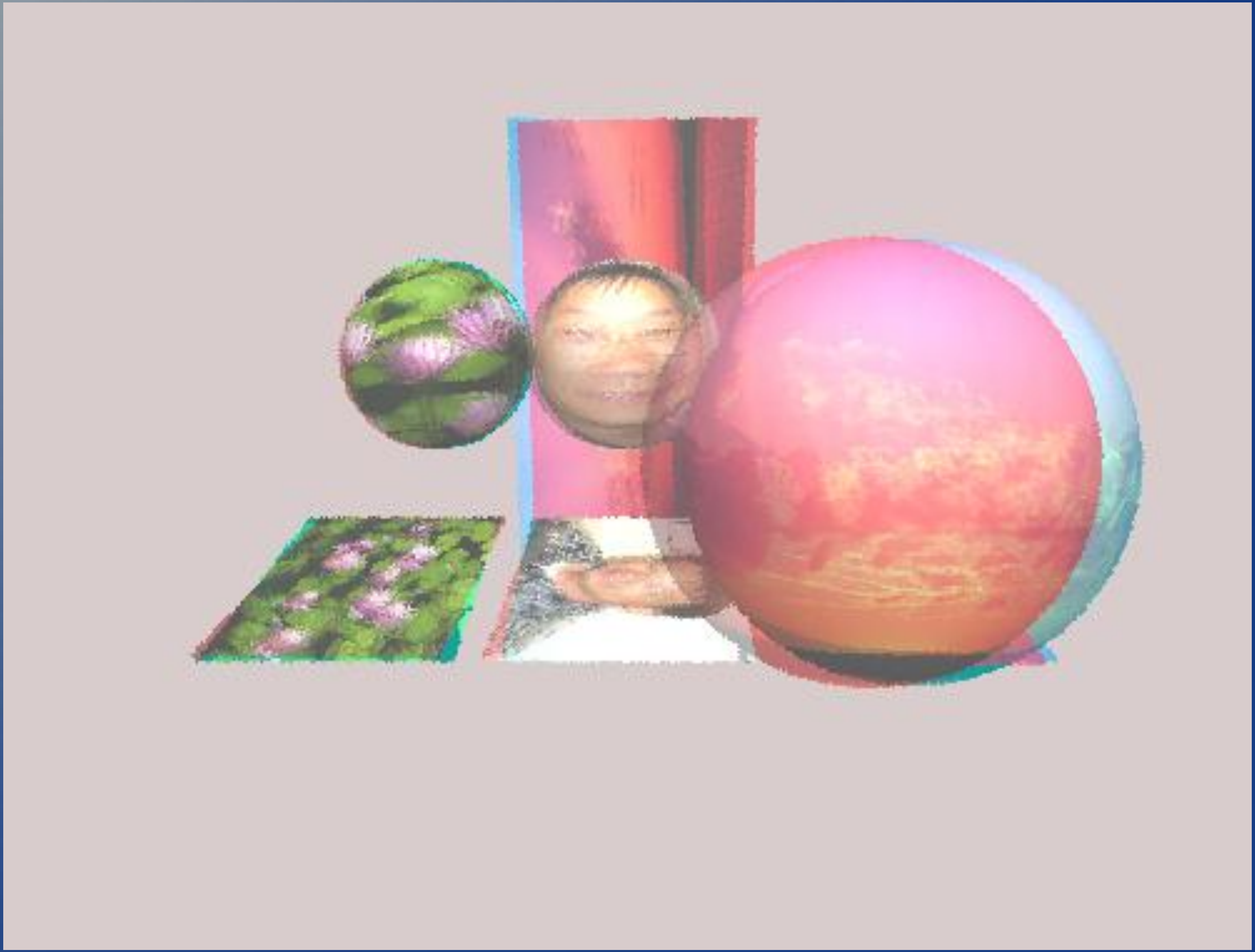

Original Right

Our Progress: emphasized images


Cyan Emphasized


Red Emphasized

Our Progress: Result



Measuring Success

The resulting image can be seen in 3D while wearing anaglyph glasses

No measuring tool. May be subjective from viewer to viewer. Eyes may have to adjust.

MESH SUPPORT

What must we know?

- What?
  - Filename
- Where?
  - Position transformation
- How?
  - Rotation transformation
  - Scale transformation
  - Material

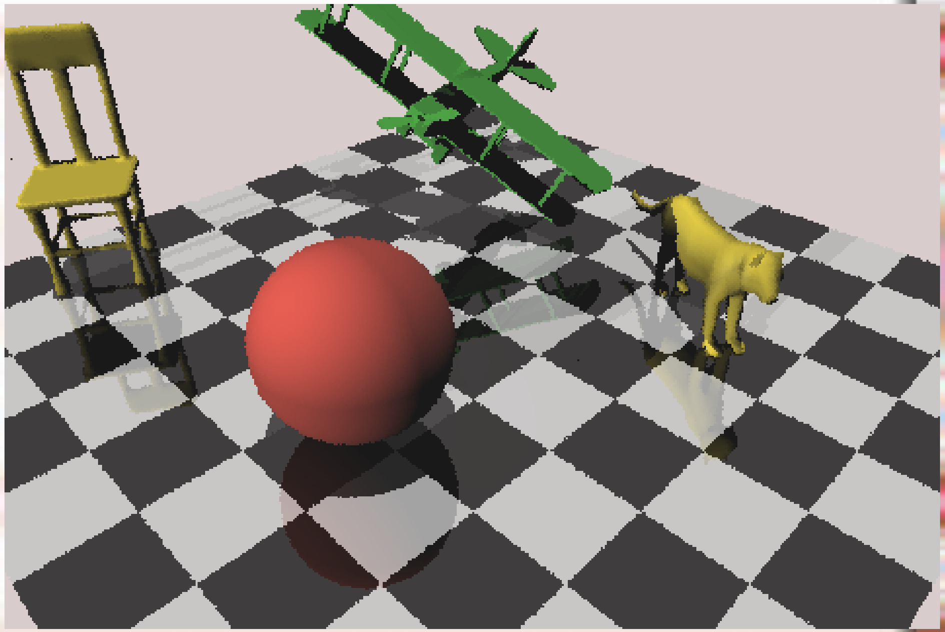
Command File

- Includes all necessary attributes
  - Filename
  - Material
  - Position, Rotation, and Scale transformations

```
<mesh>
  <filename> SphereHighPoly </filename>
  <material> 1 </material>
  <position> 10  0 20 </position>
  <rotation> 90  0 90 </rotation>
  <scale>     1  1  1 </scale>
</mesh>
```

Parsing using Graphics API

- Graphics API simplifies parsing mesh data.
- Allows us to read binary mesh files
- The API allows us to read the polygon information from the graphics pipeline.



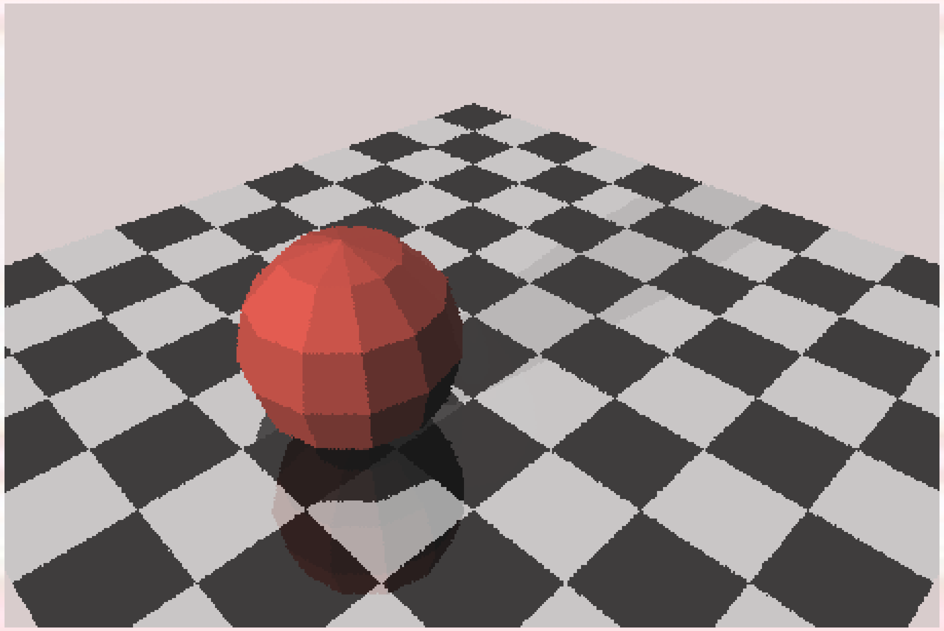
VertexBuffer and IndexBuffer

- Vertex buffer is a byte stream and contains information specified in mesh file.
  - May include: vertex position, normal vector, texture coordinates, etc.
- Index buffer is an integer stream, listing triangles by their vertices.
  - Each index corresponds to a specific vertex.
  - Vertices may be reused.

Calculating Vertex Normal

- Necessary to compute if missing from mesh
  - Phong illumination
  - Reflection

1. Calculate the normal for every triangle.
2. For every vertex, add the normal of every triangle that shares it.
3. Normalize the normal vector
   - Correct direction, wrong size.
   - Dividing by the number of additions may not result in a vector of size 1.



Ray Tracing Meshes

- Must store mesh as “Geometry” type
  - Scene compatibility
  - Intersections
- Add a Geometry_Triangle to the scene database for every polygon in the mesh.
  - Must perform world transformation!
    - Translation, rotation, scale

ACCELERATION STRUCTURES

Prerequisites

- Meshes are preloaded and respective bounding boxes calculated.
- Mesh data is parsed and triangle geometries generated.

Current Implementation

- Fastest possible time: O(S x M x N x G)
  - G = Number of geometries
  - Only G can be optimized
- Geometries are stored in a list (1D array)
- Every ray intersects every geometry in the scene.
  - Many geometries = very slow

Acceleration Structures

- Key idea: reduce number of intersection routines per ray.
- Geometries are organized in a 3D data-structure.
  - i.e., 3D array, Kd/BSP-Tree, Oct-Tree
- Best performance increase on highly-complex scenes.

Intersection Routine

1. Shoot a ray from the camera
2. Intersect ray with every cell in the data structure
   - May require entire tree traversal, depending on implementation
3. For each cell that it intersects:
   - Intersect ray with every geometry within the cell

3-dimensional array of primitives

UNIFORM SUBDIVISION MAP

Pros

- Fast
- Simple
- Traversal via line equation
  - Ray.Origin + Ray.Direction * distance

Cons

- Potentially wastes a large amount of memory for empty cells
  - Depends on geometry distribution within world
- Difficult to pick an optimal cell size
  - Large cells result in too many geometries in a single cell.
  - Small cells result in too many cells.

Determine the “world” size

- Fit tightly around geometries in world
  - Minimize void space
- Concatenate all geometry bounding boxes

```
BoundingBox world;
foreach (Geometry g in SceneDatabase)
  world = BoundingBox.CreateMerged(world,
                g.BoundBox);
```

Determine Optimal Map Size

- **Option A:** Pick a static number O(1)
  - i.e., 10x10x10, 20x20x20, 12x23x59, etc.
  - Command file vs. hardcoded constant
- **Option B:** Function of known metrics O(1)
  - i.e., SceneDatabase.Geometries.Count / world.X… Y… Z…
- **Option C:** Precompute a spatial distribution histogram O(N)

Initialize map

- Create a grid of size [x, y, z]
- Simply add all geometries that intersect each cell

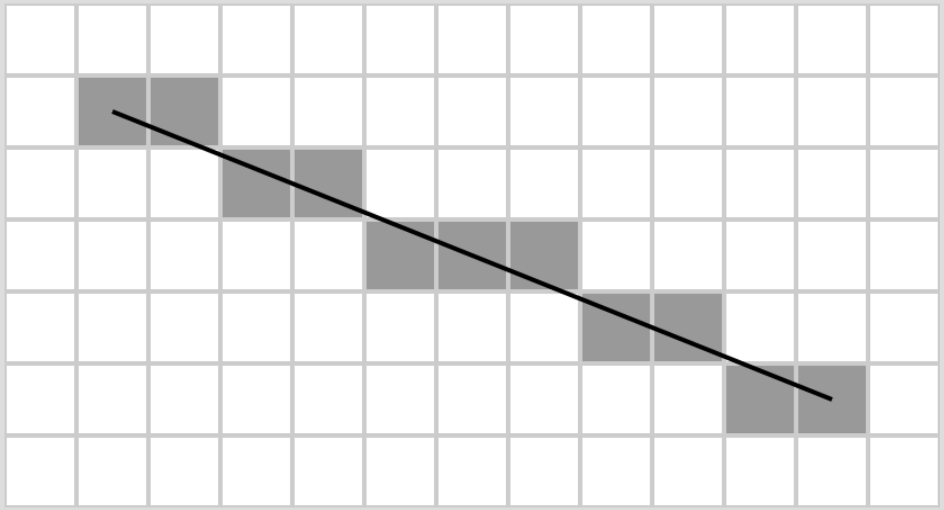
```
Cell[,,] grid = new Cell[x-size,
                         y-size,
                         z-size];

foreach (Geometry g in SceneDatabase)
  foreach (Cell c in grid)
    if (c.Intersects(g))
        c.Add(g);
```

Searching

- Follow line equation
- Perform line rasterization
- Several approaches
  - Bresenham's

- Still researching…



NON-UNIFORM SUBDIVISION MAP

Pros

- Very Fast (Goal: logn)
- Simple Tree Traversal (Usually)

Cons

- Can be complex to implement
- Takes time to initialize structure
- Difficult to choose initial partitions

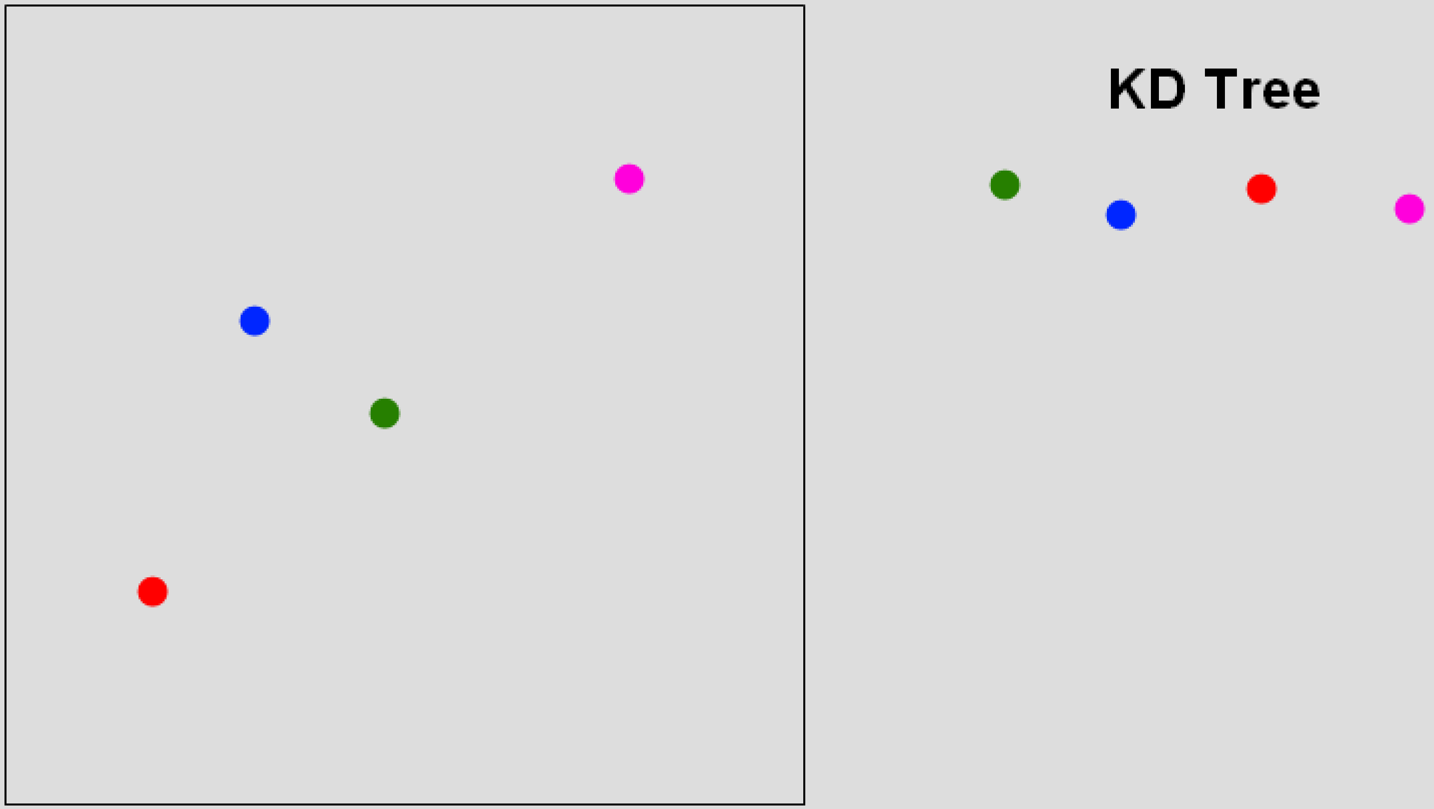
Non-Uniform Examples

- BSP Trees (Binary Space Partition Trees)
  - Divide space by arbitrary planes
- Bounding Volume Hierarchies
  - Bound geometry within volume shapes
- Bounding Interval Hierarchies
  - Super fast, super complex
  - Best of volumes and subdivision
- KD Trees
  - Special type of BSP Tree
  - Simple, effective

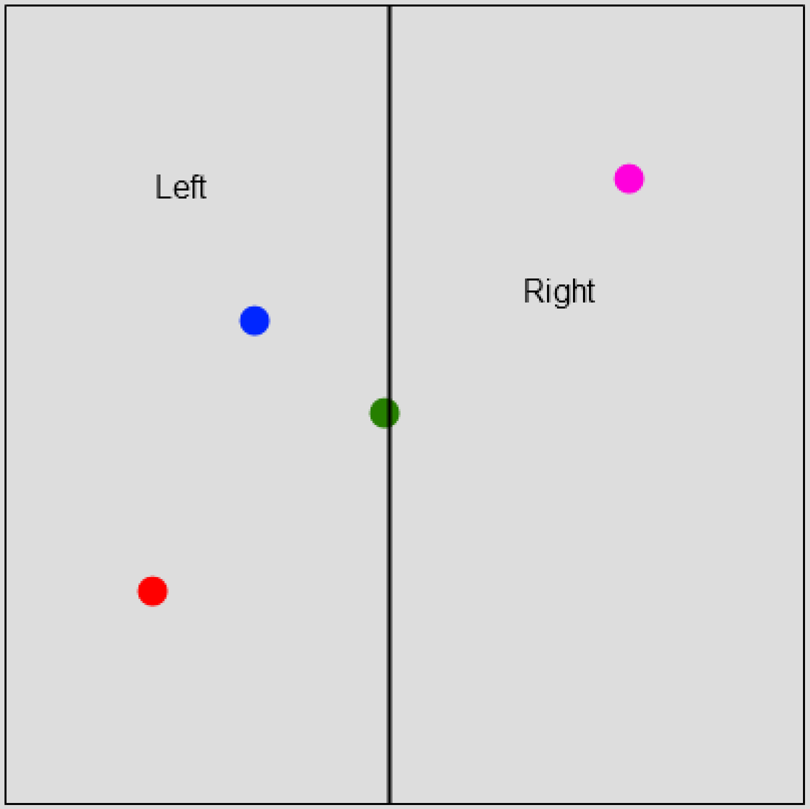
KD Trees

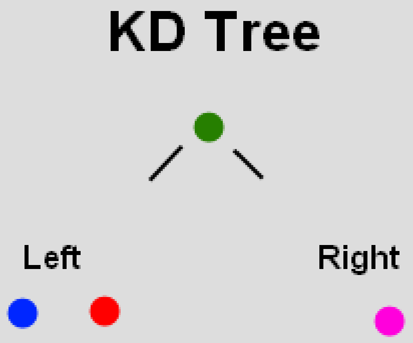
- Divide Space into Cells
- Axis-aligned Splitting Planes
  - Divide cell along one geometry
  - All other geometries fall to left or right
- Organize geometry into tree structure
- Goal: Reduce geometry
  - **n** to **logn** **(ideally—binary tree)**

KD Tree Basic Idea

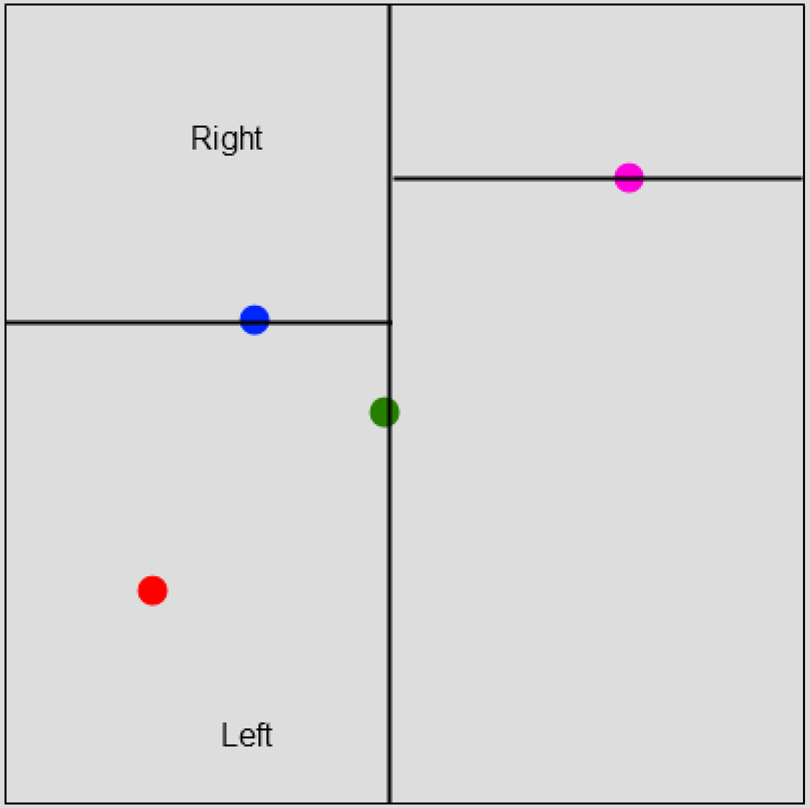


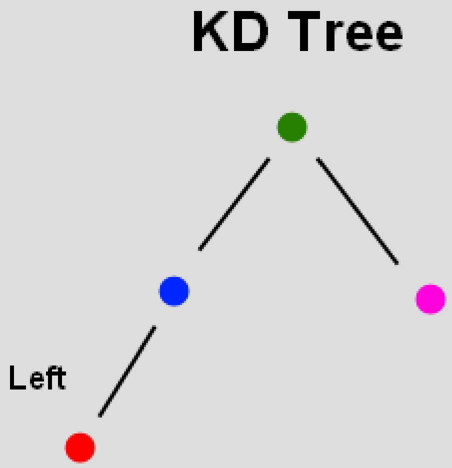
KD Tree Basic Idea



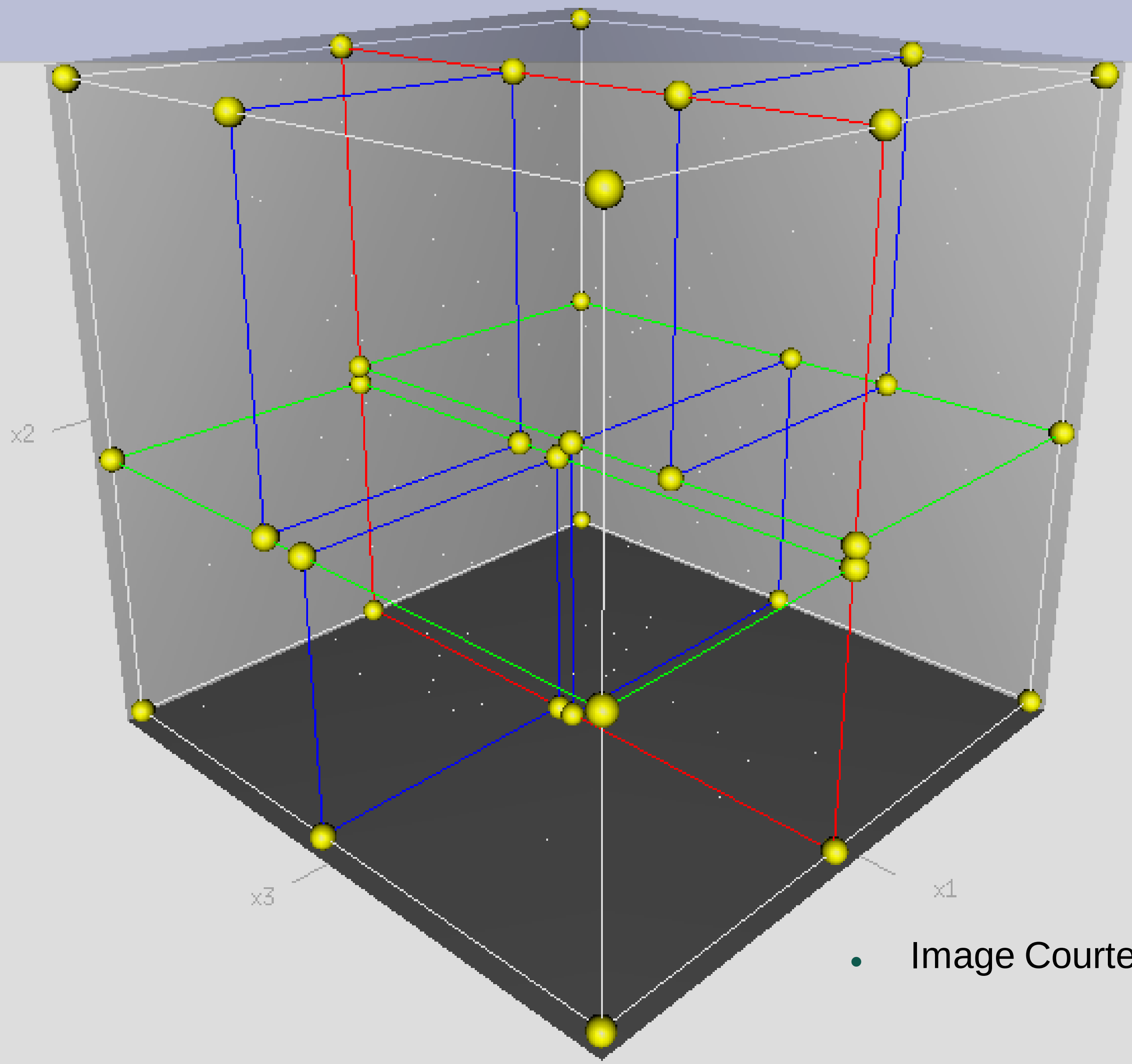


KD Tree Basic Idea



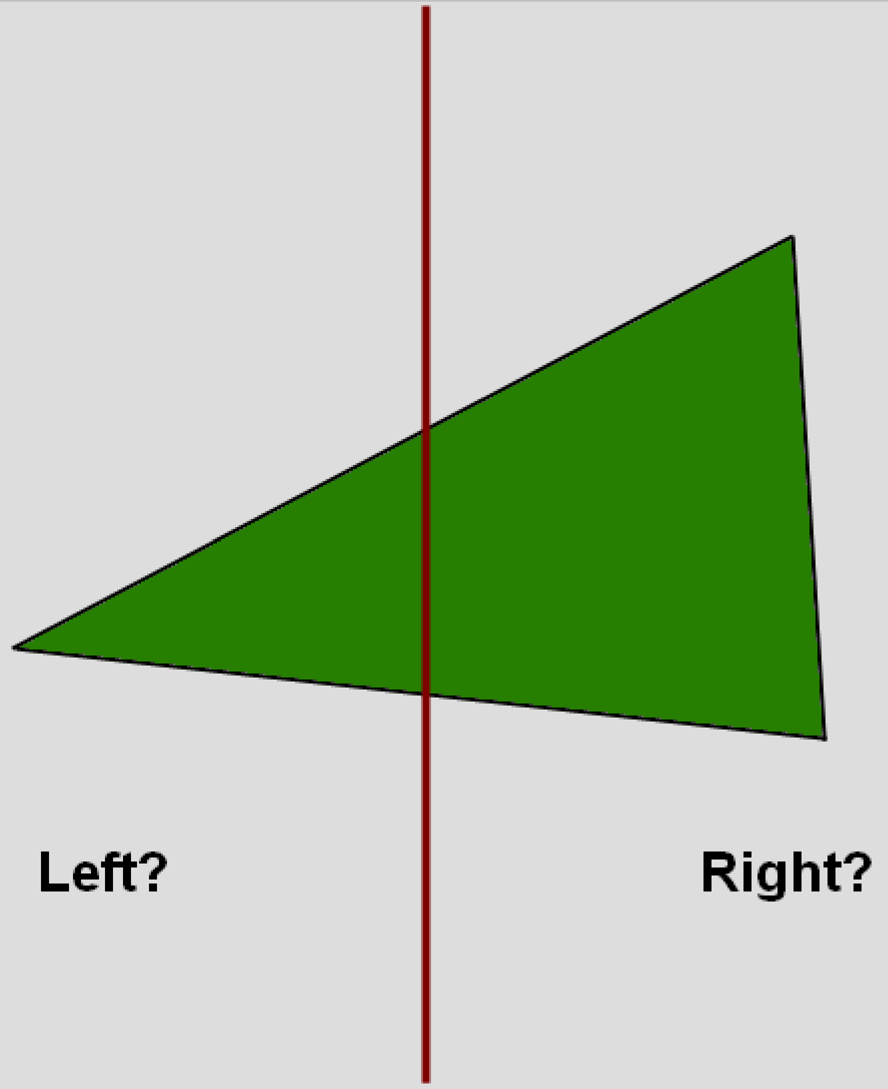


KD-Tree Spacial Subdivision

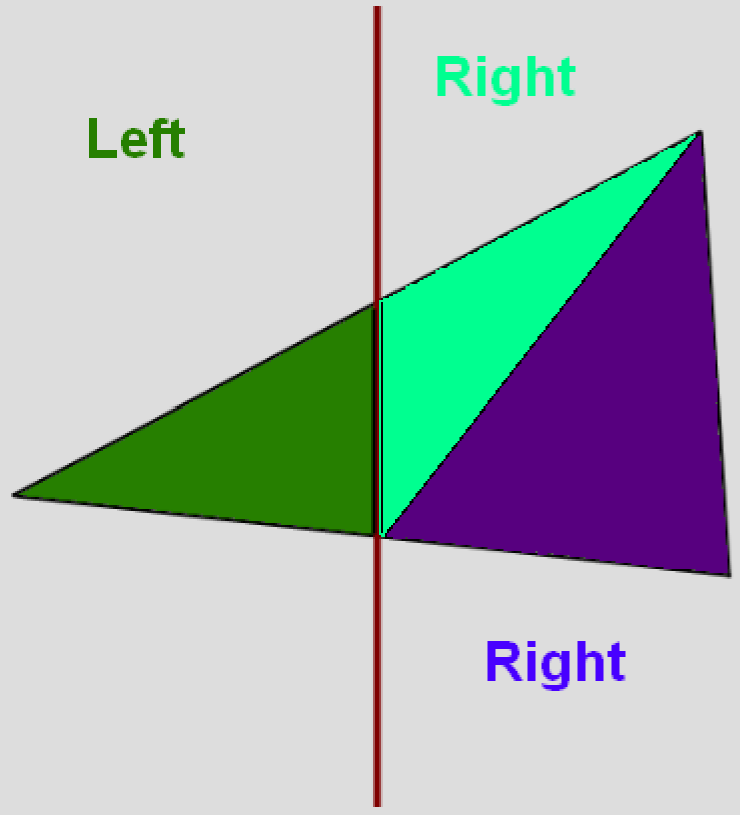


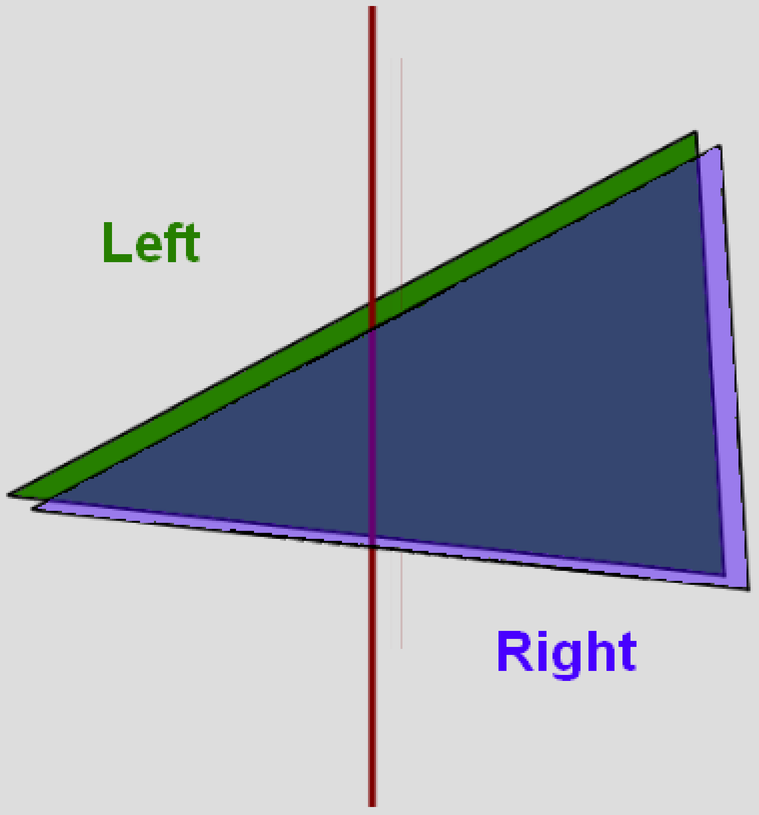
- Image Courtesy of Wikipedia

Subdivision Problem



Two Solutions





KD Tree Traversal

- Traverse Tree Structure
- Tree Position affects Intersection Priority
- Problem: Rays Cross Many Cells
- Still Being Researched

Resources I

Adventures in Ray Tracing:
  http://www.arachnoid.com/raytracing/anaglyphic_3d.html

Wikipedia:

http://en.wikipedia.org/wiki/Anaglyph_image

How to Make 3D Photos

http://www.wikihow.com/Make-3D-Photos

Resources II

- Spatial Subdivision for Ray Tracing
  http://www.cs.cmu.edu/afs/cs/project/anim-ph/463.95/pub/www/ps/spatial-subdivision.ps
- Bresenham’s Line Algorithm
  http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

Resources III

- Fast Ray Tracing using KD Trees
  http://www.cs.utexas.edu/ftp/pub/techreports/tr88-07.pdf
- KD Tree
  http://en.wikipedia.org/wiki/Kd-tree