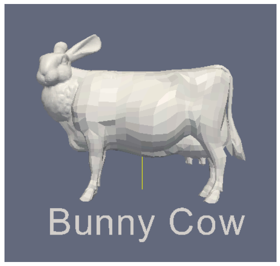
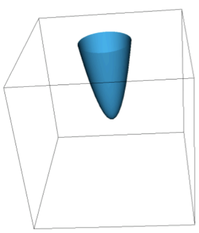
Spatial Data Structures

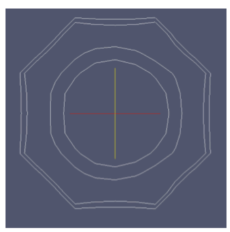


Evan Sullivan

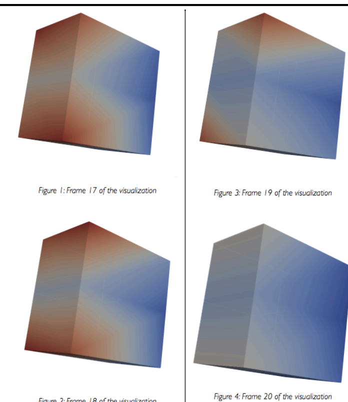
Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC

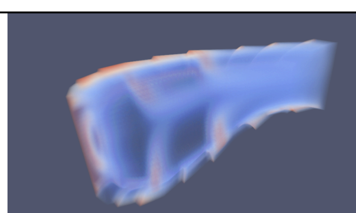


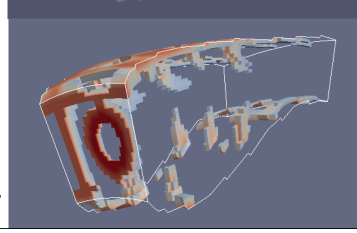


Amina Shabbeer

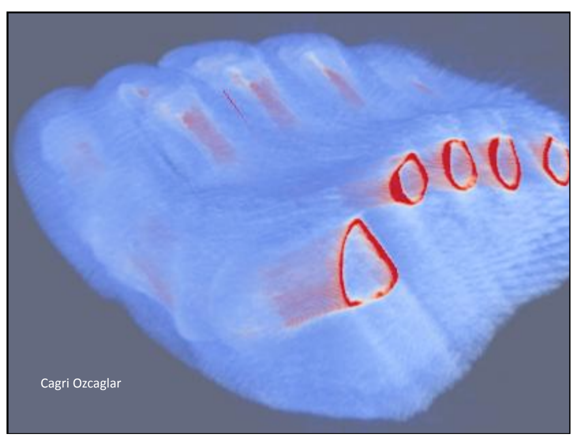


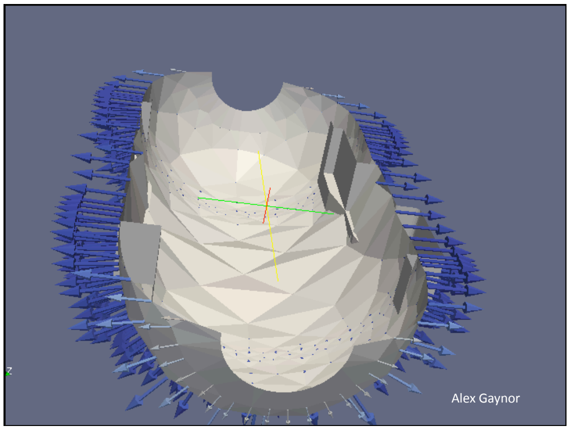
Elsa Gonsiorowski

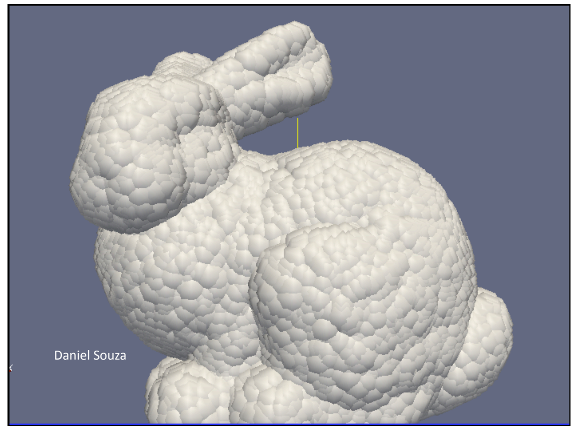


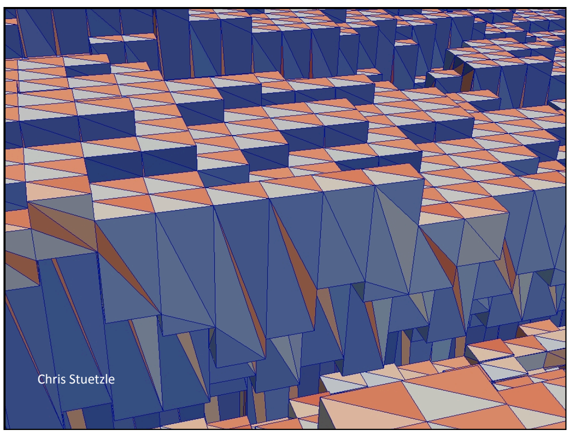


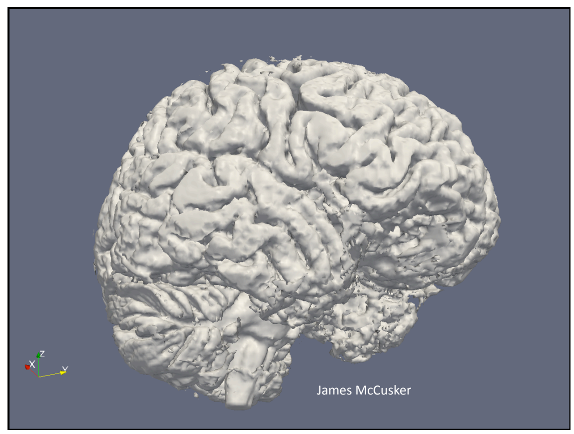
Artem Kochnev









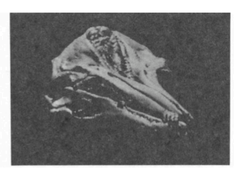


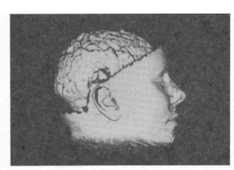
Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC

Readings for This Week:

- "A survey of algorithms for volume visualization", T. Todd Elvins, 1992





Readings for This Week:

- "Hardware-Accelerated Volume Rendering", Pfister, 2004

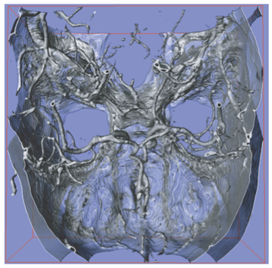


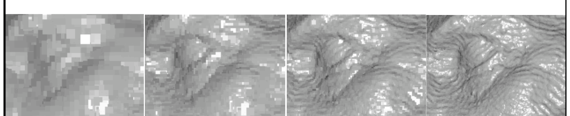
Figure 12: *CT Angiography of a human brain* ($512^2 \times 128$). *Transparent rendering of a non-polygonal shaded iso-surface with 2D multi-textures on an NVIDIA GeForce-4Ti. Image courtesy of Christof Rezk-Salama, University of Erlangen, Germany.*

Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC

Readings for Next Week:

- "QSplat: A Multiresolution Point Rendering System for Large Meshes", Rusinkiewicz & Levoy,
- SIGGRAPH 2000



Readings for Next Week:

- "Tree visualization with Tree-maps: A 2-d space-filling approach", Ben Shneiderman, 1991

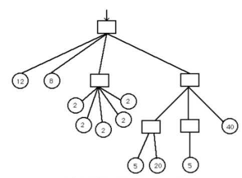


Figure 1: Typical 3-level tree structure with numbers indicating size of each leaf node

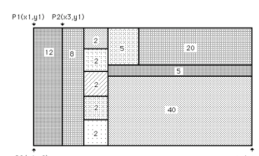


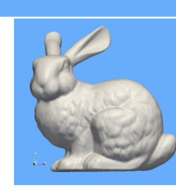
Figure 2: Tree-map of Figure 1

Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC

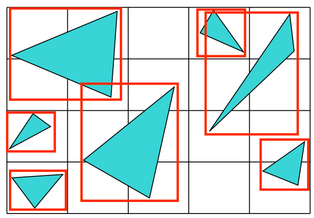
Motivation for Assignment 5

- Closest Point
  - Collision detection
  - Surface normal estimation
- Line-Polygon Intersection
  - Ray casting (& recursive ray tracing)
  - Shadow calculation
- Want to do significantly better than the linear O(n) brute force solution!



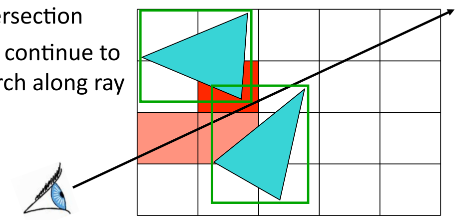
Regular Grid

- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)



For Each Cell Along a Ray

- Does the cell contain an intersection?
- Yes: return closest intersection
- No: continue to march along ray

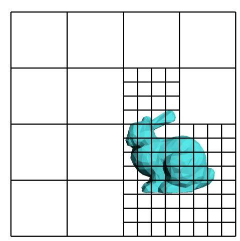


Regular Grid Discussion

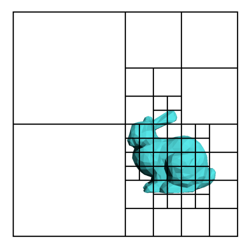
- Advantages?
  – easy to construct
  – easy to traverse
- Disadvantages?
  – may be only sparsely filled
  – geometry may still be clumped

Adaptive Grids

- Subdivide until each cell contains no more than *n* elements, or maximum depth *d* is reached



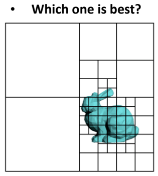
Nested Grids



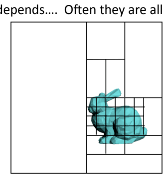
Octree/(Quadtree)

Variations of Adaptive Grids

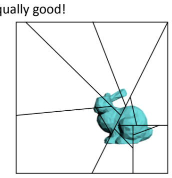
- **When to split?** When a cell contains "lots" of geometry, but has not yet reached the max tree depth
- **Where to split?**
  - Quadtree/Octree: split *every* dimension in half, always axis aligned
  - kd-tree: choose *one* dimension (often the largest dimension) and split it axis aligned (but not necessarily at the midpoint)
  - Binary Space Partition (BSP): choose a *arbitrary* cut plane
- **Which one is best?** It depends…. Often they are all equally good!



Quadtree/Octree



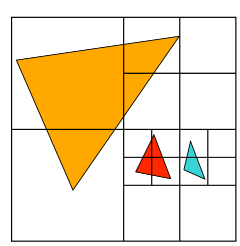
kd tree



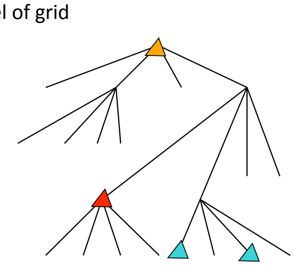
BSP Tree

Primitives in an Adaptive Grid

- Can live at intermediate levels, or be pushed to lowest level of grid

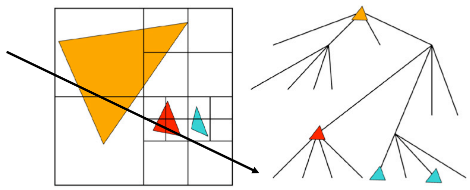


Octree/(Quadtree)



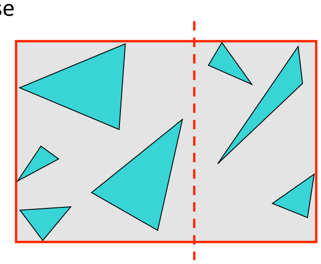
Adaptive Grid Discussion

- Advantages?
  – grid complexity matches geometric density
- Disadvantages?
  – more expensive to traverse (binary tree, lots of pointers)



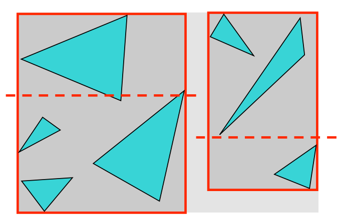
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



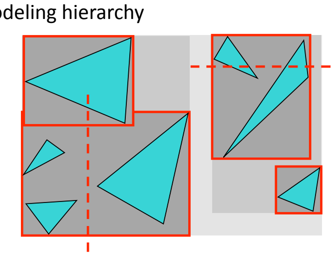
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



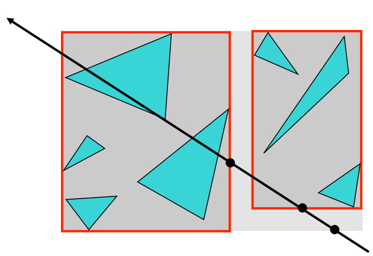
Where to split objects?

- At midpoint OR
- Sort, and put half of the objects on each side OR
- Use modeling hierarchy



Intersection with BVH

- Check sub-volume with closer intersection first

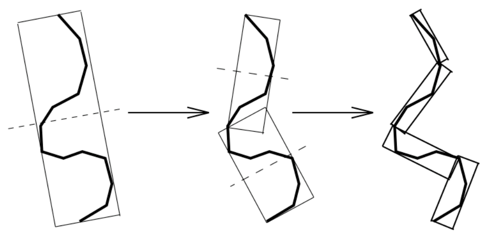


Bounding Volume Hierarchy Discussion

- Advantages
  – easy to construct
  – easy to traverse
  – binary
- Disadvantages
  – may be difficult to choose a good split for a node
  – poor split may result in minimal spatial pruning

Oriented Bounding Box (OBB)

- Generalization of the (axis-aligned) BVH



OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, Gottschalk, Lin, & Manocha, SIGGRAPH 1996.

Teams for Assignment 5

- KD Tree: James, Amina, Tyler
- Octree: Greg, Mary, Nick
- Oriented Bounding Box (OBB) Tree: Chris, Andrew, Jonathan, Cagri,
- Modified BSP Tree: Evan, Dan, Will, Elsa

Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC

Introduction to VTK: Spatial Data Structures

Spatial Data Structures

- Efficient computations
- Versus iterating over an entire large data set
- Operations include:
  - Closest point
  - Line-polygon intersection

Choices in VTK

- Octree
- Kd-Tree
- OBB (Oriented Bounding Box) Tree
- Modified BSP (Binary Space Partition) Tree

How to make the right choice?

- Lucky

- Experimentation

- Magic

VTK's Implementations

	Closest Point	Line Intersection
KD Tree	yes	no
Octree	yes	no
OBB Tree	no	yes
Modified BSP Tree	no	yes

Octree

```
// Build the tree
vtkOctreePointLocator octree =
  vtkOctreePointLocator::New();
octree->SetDataSet(polydata);
octree->BuildLocator();

// Perform a query
double testPoint[3] = {2.0, 0.0, 0.0};
vtkIdType id = octree->FindClosestPoint(testPoint);
cout << "The closest point is point " << id << endl;
```

Kd-Tree

```
// Build the tree
vtkKdTreePointLocator kDTree =
  vtkKdTreePointLocator::New();
kDTree->SetDataSet(polydata);
kDTree->BuildLocator();

// Perform a query
double testPoint[3] = {2.0, 0.0, 0.0};
vtkIdType id = kDTree->FindClosestPoint(testPoint);
cout << "The closest point is point " << id << endl;
```

OBB Tree

```
// Build the tree
vtkOBBTree* tree = vtkOBBTree::New();
tree->SetDataSet(polydata);
tree->BuildLocator();

// Intersect the locator with the line
double lineP0[3] = {0.0, 0.0, 0.0};
double lineP1[3] = {0.0, 0.0, 2.0};
vtkPoints* intersectPoints =
  vtkPoints::New();
tree->IntersectWithLine(lineP0, lineP1, intersectPoints, NULL);
double intersection[3];
intersectPoints->GetPoint(0, intersection);
cout << "NumPoints: " << intersectPoints->GetNumberOfPoints() << endl;
cout << "Intersection: " << intersection[0] << ", " << intersection[1] << ", " <<
intersection[2] << endl;
```

Modified BSP Tree

```
// Build the tree
vtkModifiedBSPTree* bspTree =
  vtkModifiedBSPTree::New();
bspTree->SetDataSet(polydata);
bspTree->BuildLocator();

// Setup a line
double p1[3] = {-2,0,0};
double p2[3] = {2,0,0};
double tolerance = .001;
```

Modified BSP Tree (cont.)

```
// Outputs
double t;
// Parametric coordinate of intersection (0 (corresponding to p1) to 1
(corresponding to p2))

double x[3]; // The coordinate of the intersection

double pcoords[3];
int subId;
vtkIdType iD = bspTree->IntersectWithLine(p1, p2, tolerance, t, x, pcoords, subId);

cout << "t: " << t << endl;
cout << "x: " << x[0] << " " << x[1] << " " << x[2] << endl;
```

Today's Class

- Highlights from HW #4
- This Week's Readings
- Next Week's Readings
- Spatial Data Structures
- Data Structures in VTK
- Final Projects & Arts & EMPAC
  - Next Week: joint meeting with Kathleen Ruiz's "Advanced Integrated Arts" course
  - Immersive Interactive Environments in EMPAC

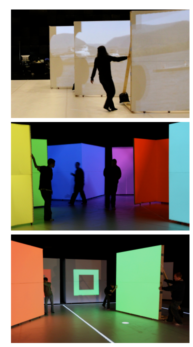
Dynamic Projection Surfaces for Immersive Visualization

Theodore C. Yapo, Yu Sheng, Joshua Nasman, Andrew Dolce, Eric Li, and Barbara Cutler

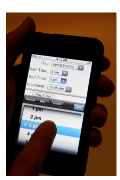
PROCAMS 2010 IEEE International Workshop on Projector-Camera Systems, June 2010

Our System Goals/Requirements

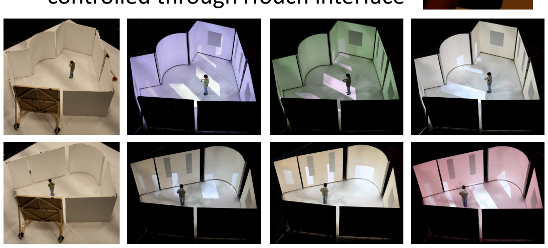
- Large, human-scale projection environment
- People move freely within the space
- Projection surfaces can be moved interactively
- Varying illumination conditions
- Robust & real-time tracking and display



Architectural Daylighting Design

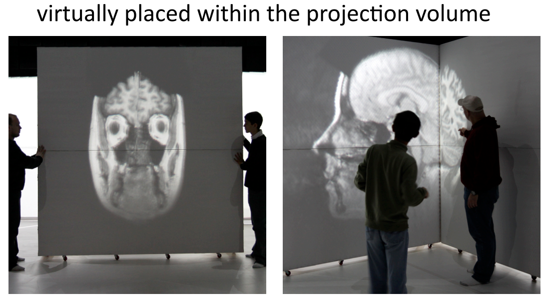


- Windows, wall colors, & time of day controlled through iTouch interface



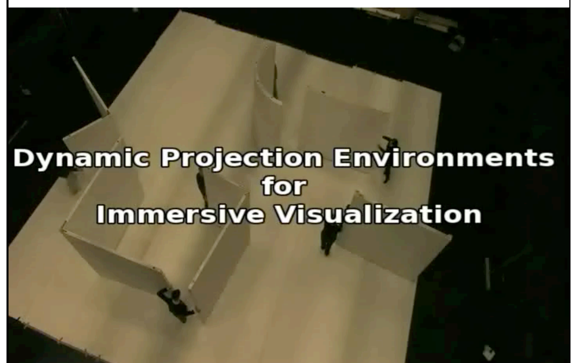
Volumetric Visualization

- Cross sections of a 3D medical dataset virtually placed within the projection volume



General User Interface Elements

- Projection surfaces as input devices
- No instruction necessary to play the game!



Panorama from Gehua Yang, DualAlign