Geometric Data Structures for Computer Graphics

Dr. Gabriel Zachmann

Dr. Elmar Langetepe

University Bonn

Germany

{zach,langetep}@cs.uni-bonn.de

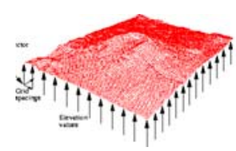
Introduction

- What this tutorial is about
- What it is *not* about

Overview

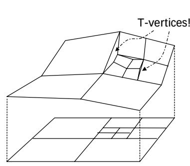
Terrain Visualization

- Problem
  - Given: height values on regular 2D grid
  - Task: render with 60 Hz
- Brute-force solution
  - Render ~ 500 Mio tris
- Better solution
  - view-dependent dyn. LOD, stripes, cache locality
- Idea: Quadtrees



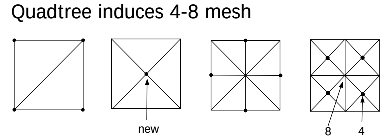
Avoiding Cracks

- Cannot render quadrangles
  - Probably not planar
  - Cracks because of T-vertices
- Must render triangles

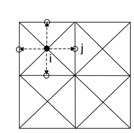


Subdivision Scheme

- Quadtree induces 4-8 mesh



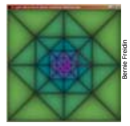
- Induces DAG
  - "vertex j is child of i" ⇔ j is created by splitting at i
  - Denote this by an edge (i,j)



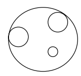
Dependency among Triangles

- Graph-theoretic condition

  Let M^0 be the complete DAG,
  let M be a sub-graph of M^0.
  M yields a crack-free terrain ⇔
  $\forall j \in M : (i, j) \in M^0 \Rightarrow (i, j) \in M$

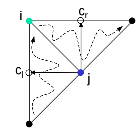


- Rendering condition:
  - Find criterion for vertices that has the "nesting property":
    criterion(j) = "render it" ⇒
    ∀ parents i: criterion(i) = "render it"



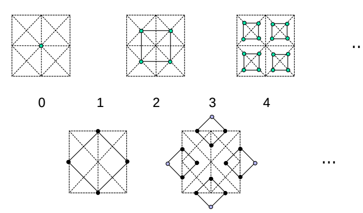
Procedure for Rendering

<u>submesh(i,j)</u>
if error(i) < τ **then**
  return
if B_i outside view **then**
  return
submesh(j,c_l)
V += p_i
submesh(j,c_r)

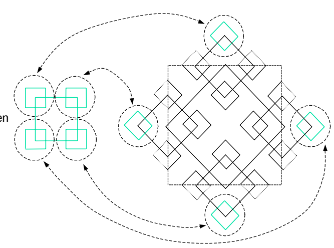


Storing the Quadtree

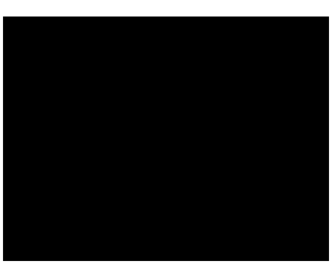
- Don't use pointers
- Find numbering scheme with little "dead numbers"
- Observation: subdivision scheme induces 2 quadtrees

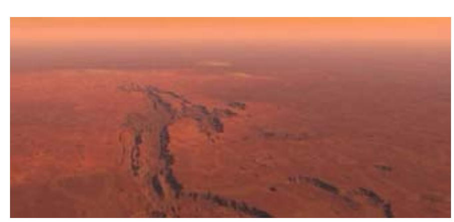


- Storing the "green" quadtree in the "black" one:



Movies

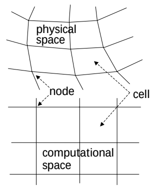




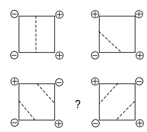
NASA

Isosurface Generation

- Problem
  - Given: scalar field $f : \mathbb{R}^3 \to \mathbb{R}$
  - Task: find polygonal repr. of $f(\mathbf{x}) = t$
  - Discrete: curvilinear grid / regular grid
  - Space: physical / computational space
  - Task (discrete): find all cells with a node < t and a node > t

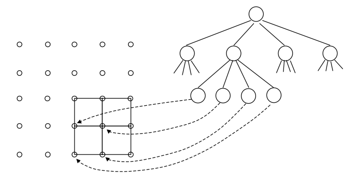


- Simple algo ("marching cubes")
  1. Compute sign for all nodes (⊕ = >t);
  2. Triangulate all cells according to LUT



Octrees over Volume Data

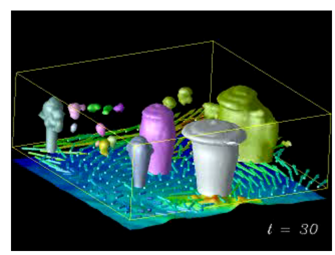
- Leaf: ptr to lower left node
- Inner node: ptr to first child
- All nodes ν store ν_{min} and ν_{max}



Isosurface Generation with Octree

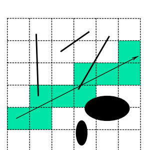
- Isosurface intersects volume assoc. with node ν $\Leftrightarrow \nu_{min} < t < \nu_{max}$
- Algo (obvious)
  - Start with root
  - Recurse into nodes satisfying condition
- Improvement
  - Observation: edges are visited exactly 4 times
  - Keep hash table of edges

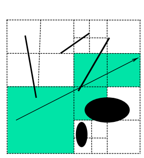
Movie



Ray Shooting

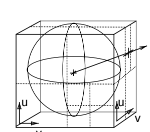
- Applications: ray tracing, radiosity, volume visualization, terrain following, etc.
- Simplest solution: grid
- 3D octree
  - Bottom-up
  - Top-down

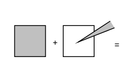


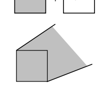


5D Octree for Rays

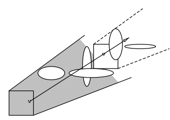
- What is a ray?
  - Point + direction = 5-dim. Object
- Octree over rays
  - "Direction cube"
  - One-to-one mapping for dir's: $S^2 \leftrightarrow D := [-1,+1]^2 \times \{\pm x, \pm y, \pm z\}$
  - All rays in universe $U = [0,1]^3$ $R = U \times D$
- Node of 5D octree = beam in 3D







- Construction
  - Start with root node $= U \times [-1,+1]^2$ and all objects associated
  - Partition node iff
    1. Too many objects, *and*
    2. Cell too large.
    - Partition set of objcets
- Shooting rays
  1. Convert ray to 5D point
  2. Find leaf of octree
  3. Intersect ray with associated objects
- Optimizations ...

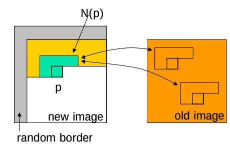


Texture Synthesis

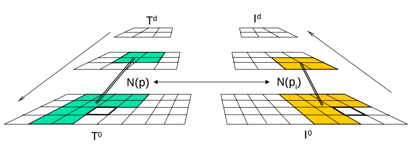
- Properties of textures
  - Stationary under moving window
  - Locality of dependency

- Algorithm

for all p $\in$ new image **do**
find $p_i \in$ old image so that
$|N(p) - N(p_i)|^2 = \min$
set p := p_i



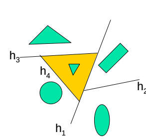
- Better independence from size of N(p)

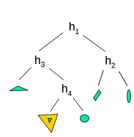


- Examples

BSP Trees

- Generalization of k-d trees
- Definition (recursive)
  - S = set of objects,
    S(ν) = objects assoc. with node ν,
    T(S) = BSP for set S
  1. Case |S|≤1:
     T = leaf ν storing $S(\nu) := S$
  2. Case |S|≥1:
     T = tree with root ν storing h_ν and $S(\nu)$,
     $S(\nu) := \{x \in S \mid x \subseteq h_\nu\}$
     children for sets $S^+(\nu)$ and $S^-(\nu)$,
     $S^+(\nu) := \{x \cap h_\nu^+ \mid x \in S\}$

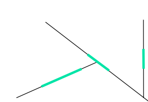


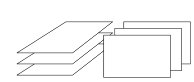


Autopartitions

- Properties
  - Each h_ν = plane of one polygon
  - Each S(ν) = that polygon
- Complexity

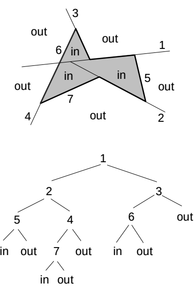
  $O(n \log n)$
  - In 2D: proven
  - In 3D: experience for "well-behaved" geometry





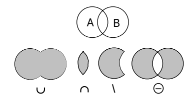
BSPs for Object Representation

- Difference to orig. definition: stop only when |S|=0
- Leaves
  - Homogenous convex cells
  - Either inside or outside
- Construction
  - Guided by heuristic

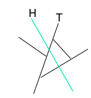


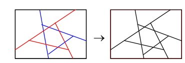
Boolean Operations

- Operations: $\cap$ $\cup$ \ $\ominus$



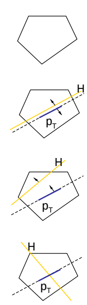
- Algorithm
  1. Split BSP by plane
  2. Merge two BSPs
  3. Compute operation on cells





Subalgorithm 1

- Split BSP T by plane H, polygon p at root of T
- Output two new BSPs
- Cases:
  1. T is leaf: trivial ...
  2. $p \subset H$: return children
  3. H completely on one side of p: split one child, combine with other child
  4. H crosses p: split both children, recombine across p

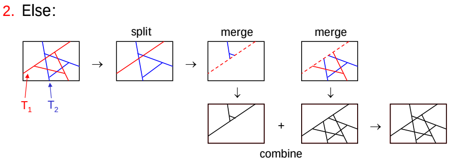


Subalgorithm 2

- Merge T_1 and T_2
- Output T with leaf cells c such that

$$C = \{c \mid c = c_1 \cap c_2, c_1 \in C_1, c_2 \in C_2\}$$

- Algorithm
  1. T_1 or T_2 is leaf: perform operation on cell
  2. Else:



Subalgorithm 3

- The Cell Operation

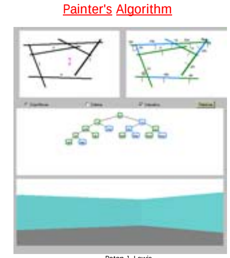
Op	T_1	Result
$\cup$	in	T_1
	out	T_2
$\cap$	in	T_2
	out	T_1
\	in	T_2^C
	out	T_1
$\ominus$	in	T_2^C
	out	T_2

Demos

Boolen Operations

Stan Melax

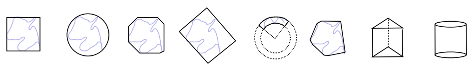
Painter's Algorithm



Paton J. Lewis

Bounding Volume Hierarchies

- Definition (informal):
  - Tree, nodes carry BV
  - Leaves carry one (or more) "primitives"
  - BV of node contains BVs of all children
  - Leaf BV contains primitive
- Many variables
- Bounding Volumes



- Tightness

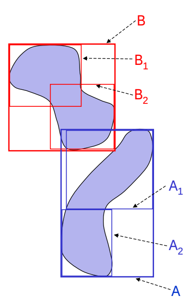
- Applications
  - Ray shooting
  - Nearest-neighbor
  - Frustum and occlusion culling
  - Geographical data bases
  - Collision detection
- Construction
  - Strategies
    - Bottom-up
    - Insertion
    - Top-down
  - Heuristic!

Interactive hierarchy construction

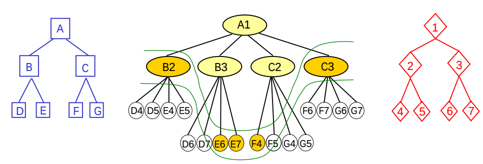
Collision Detection

- Simultaneous traversal:

<u>traverse(A,B)</u>
if A,B do not overlap **then**
  return
if A and B are leaves **then**
  check primitives
else
  forall children A_i, B_j **do**
    traverse(A_i, B_j)



- The recursion tree (what the algo really does):



Movies

Remaining primitives

A simple application

Thanks Folks

A Continuum of Data Structures