



# Ray Tracing

***Wolfgang Heidrich***

© Wolfgang Heidrich



## Course News

### ***Assignment 3 (project)***

- Due April 1
- Demos in labs April 2-7

### ***Reading***

- Chapter 10 (ray tracing), except 10.8-10.10
- Chapter 14 (global illumination)

Wolfgang Heidrich

## Course Topics for the Rest of the Term



### **Ray-tracing & Global Illumination**

- This week

### **Parametric Curves/Surfaces**

- March 30/April 1
- Taught by Robert Bridson - I will be at a conference

### **Overview of current research**

- April 3/6 (Ivo Ihrke – I am still at conference)

### **April 8 – Final Q&A (I will be back for that)**

Wolfgang Heidrich

## Ray-Tracing



### **Basic Algorithm (Whithead):**

```
for every pixel  $p_i$  {  
  Generate ray  $r$  from camera position through pixel  $p_i$   
   $p_i$  = background color  
  for every object  $o$  in scene {  
    if(  $r$  intersects  $o$  && intersection is closer than  
      previously found intersections )  
      Compute lighting at intersection point, using local  
      normal and material properties; store result in  $p_i$   
  }  
}
```

Wolfgang Heidrich



## Ray-Tracing

### Issues:

- Generation of rays
- Intersection of rays with geometric primitives
- Geometric transformations
- Lighting and shading
- Efficient data structures so we don't have to test intersection with every object

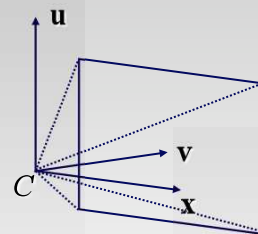
Wolfgang Heidrich



## Ray-Tracing – Generation of Rays

### Camera Coordinate System

- Origin: C (camera position)
- Viewing direction:  $\mathbf{v}$
- Up vector:  $\mathbf{u}$
- x direction:  $\mathbf{x} = \mathbf{v} \times \mathbf{u}$



### Note:

- Corresponds to viewing transformation in rendering pipeline!
- See gluLookAt...

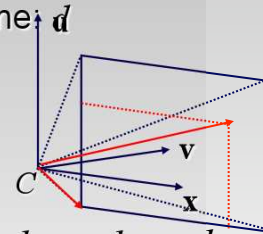
Wolfgang Heidrich

## Ray-Tracing – Generation of Rays



### Other parameters:

- Distance of Camera from image plane:  $d$
- Image resolution (in pixels):  $w, h$
- Left, right, top, bottom boundaries in image plane:  $l, r, t, b$



### Then:

- Lower left corner of image:  $O = C + d \cdot \mathbf{v} + l \cdot \mathbf{x} + b \cdot \mathbf{u}$
- Pixel at position  $i, j$  ( $i=0..w-1, j=0..h-1$ ):

$$P_{i,j} = O + i \cdot \frac{r-l}{w-1} \cdot \mathbf{x} - j \cdot \frac{t-b}{h-1} \cdot \mathbf{u}$$

$$= O + i \cdot \Delta x \cdot \mathbf{x} - j \cdot \Delta y \cdot \mathbf{y}$$

Wolfgang Heidrich

## Ray-Tracing – Generation of Rays



### Ray in 3D Space:

$$R_{i,j}(t) = C + t \cdot (P_{i,j} - C) = C + t \cdot \mathbf{v}_{i,j}$$

where  $t = 0 \dots \infty$

Wolfgang Heidrich



## Ray-Tracing

### Issues:

- Generation of rays
- **Intersection of rays with geometric primitives**
- Geometric transformations
- Lighting and shading
- Efficient data structures so we don't have to test intersection with every object

Wolfgang Heidrich



## Ray Intersections

### Task:

- Given an object  $o$ , find ray parameter  $t$ , such that  $\mathbf{R}_{i,j}(t)$  is a point on the object
  - *Such a value for  $t$  may not exist*
- Intersection test depends on geometric primitive

Wolfgang Heidrich



## Ray Intersections

### Spheres at origin:

- Implicit function:

$$S(x, y, z) : x^2 + y^2 + z^2 = r^2$$

- Ray equation:

$$\mathbf{R}_{i,j}(t) = \mathbf{C} + t \cdot \mathbf{v}_{i,j} = \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} + t \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} c_x + t \cdot v_x \\ c_y + t \cdot v_y \\ c_z + t \cdot v_z \end{pmatrix}$$

Wolfgang Heidrich



## Ray Intersections

### To determine intersection:

- Insert ray  $\mathbf{R}_{i,j}(t)$  into  $S(x, y, z)$ :

$$(c_x + t \cdot v_x)^2 + (c_y + t \cdot v_y)^2 + (c_z + t \cdot v_z)^2 = r^2$$

- Solve for  $t$  (find roots)
  - Simple quadratic equation

Wolfgang Heidrich



## Ray Intersections

### Other Primitives:

- Implicit functions:
  - Spheres at arbitrary positions
    - Same thing
  - Conic sections (hyperboloids, ellipsoids, paraboloids, cones, cylinders)
    - Same thing (all are quadratic functions!)
  - Higher order functions (e.g. tori and other quartic functions)
    - In principle the same
    - But root-finding difficult
    - Not to resolve to numerical methods

Wolfgang Heidrich



## Ray Intersections

### Other Primitives (cont)

- Polygons:
  - First intersect ray with plane
    - linear implicit function
  - Then test whether point is inside or outside of polygon (2D test)
  - For convex polygons
    - Suffices to test whether point in on the right side of every boundary edge
    - Similar to computation of outcodes in line clipping

Wolfgang Heidrich



## Ray-Tracing

### **Issues:**

- Generation of rays
- Intersection of rays with geometric primitives
- **Geometric transformations**
- Lighting and shading
- Efficient data structures so we don't have to test intersection with every object

Wolfgang Heidrich



## Ray-Tracing – Geometric Transformations

### **Geometric Transformations:**

- Similar goal as in rendering pipeline:
  - *Modeling scenes convenient using different coordinate systems for individual objects*
- Problem:
  - *Not all object representations are easy to transform*
    - This problem is fixed in rendering pipeline by restriction to polygons (affine invariance!)

Wolfgang Heidrich



## Ray-Tracing – Geometric Transformations



### **Geometric Transformations:**

- Similar goal as in rendering pipeline:
  - *Modeling scenes convenient using different coordinate systems for individual objects*
- Problem:
  - *Not all object representations are easy to transform*
    - This problem is fixed in rendering pipeline by restriction to polygons (affine invariance!)
  - *Ray-Tracing has different solution:*
    - The ray itself is always affine invariant!
    - Thus: transform ray into object coordinates!

Wolfgang Heidrich

## Ray-Tracing – Geometric Transformations



### **Ray Transformation:**

- For intersection test, it is only important that ray is in same coordinate system as object representation
- Transform all rays into object coordinates
  - *Transform camera point and ray direction by inverse of model/view matrix*
- Shading has to be done in world coordinates (where light sources are given)
  - *Transform object space intersection point to world coordinates*
  - *Thus have to keep both world and object-space ray*

Wolfgang Heidrich



## Ray-Tracing

### **Issues:**

- Generation of rays
- Intersection of rays with geometric primitives
- Geometric transformations
- **Lighting and shading**
- Efficient data structures so we don't have to test intersection with every object

Wolfgang Heidrich



## Ray-Tracing Lighting and Shading

### **Local Effects:**

- Local Lighting
  - *Any reflection model possible*
  - *Have to talk about light sources, normals...*
- Texture mapping
  - *Color textures*
  - *Bump maps*
  - *Environment maps*
  - *Shadow maps*

Wolfgang Heidrich

# Ray-Tracing Local Lighting



## Light sources:

- For the moment: point and directional lights
- Later: are light sources
- More complex lights are possible
  - Area lights
  - Global illumination
    - Other objects in the scene reflect light
    - Everything is a light source!
    - Talk about this on Monday

Wolfgang Heidrich

# Ray-Tracing Local Lighting



## Local surface information (normal...)

- For implicit surfaces  $F(x,y,z)=0$ : normal  $\mathbf{n}(x,y,z)$  can be easily computed at every intersection point using the gradient

$$\mathbf{n}(x, y, z) = \begin{pmatrix} \partial F(x, y, z) / \partial x \\ \partial F(x, y, z) / \partial y \\ \partial F(x, y, z) / \partial z \end{pmatrix}$$

- Example:  $F(x, y, z) = x^2 + y^2 + z^2 - r^2$

$$\mathbf{n}(x, y, z) = \begin{pmatrix} 2x \\ 2y \\ 2z \end{pmatrix} \quad \text{Needs to be normalized!}$$

Wolfgang Heidrich

# Ray-Tracing Local Lighting



## Local surface information

- Alternatively: can interpolate per-vertex information for triangles/meshes as in rendering pipeline
  - Phong shading!
  - Same as discussed for rendering pipeline
- Difference to rendering pipeline:
  - Interpolation cannot be done incrementally
  - Have to compute Barycentric coordinates for every intersection point (e.g plane equation for triangles)

Wolfgang Heidrich

# Ray-Tracing Texture Mapping



## Approach:

- Works in principle like in rendering pipeline
  - Given  $s, t$  parameter values, perform texture lookup
  - Magnification, minification just as discussed
- Problem: how to get  $s, t$ 
  - Implicit surfaces often don't have parameterization
  - For special cases (spheres, other conic sections), can use parametric representation
  - Triangles/meshes: use interpolation from vertices

Wolfgang Heidrich

# Ray-Tracing Lighting and Shading



## Global Effects

- Shadows
- Reflections/refractions

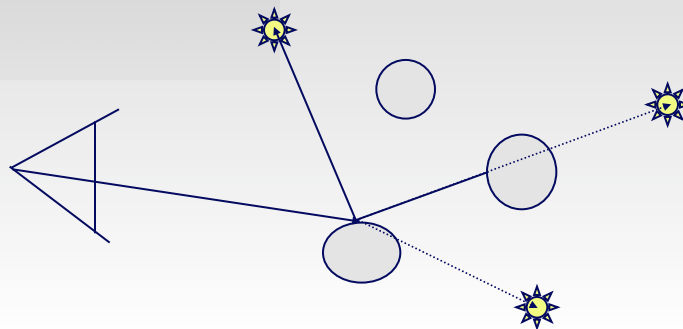
Wolfgang Heidrich

# Ray-Tracing Shadows



## Approach:

- To test whether point is in shadow, send out shadow rays to all light sources
  - If ray hits another object, the point lies in shadow



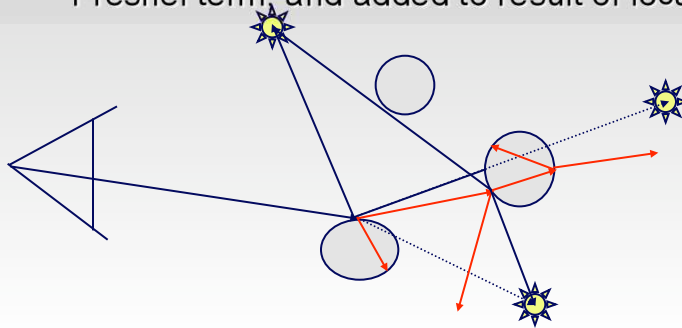
Wolfgang Heidrich

# Ray-Tracing Reflections/Refractions



## Approach:

- Send rays out in reflected and refracted direction to gather incoming light
- That light is multiplied by local surface color and Fresnel term, and added to result of local shading



Wolfgang Heidrich

# Recursive Ray Tracing

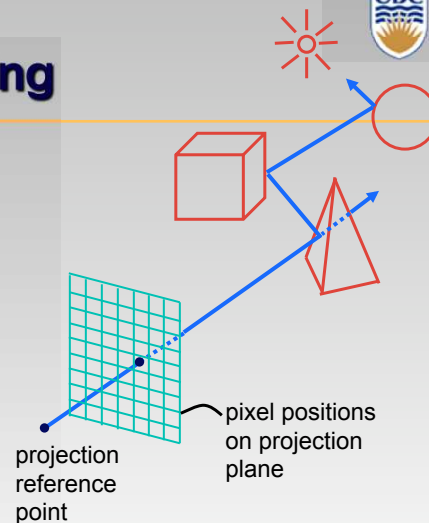


## Ray tracing can handle

- Reflection (chrome)
- Refraction (glass)
- Shadows

## Spawn secondary rays

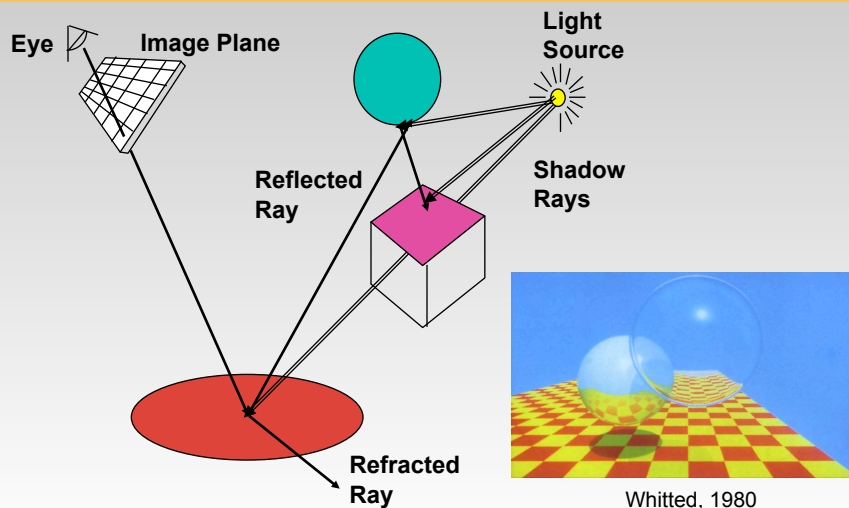
- Reflection, refraction
  - If another object is hit, recurse to find its color
- Shadow
  - Cast ray from intersection point to light source, check if intersects another object



Wolfgang Heidrich



## Recursive Ray-Tracing



Wolfgang Heidrich



## Recursive Ray-Tracing Algorithm

```
RayTrace(r,scene)
obj := FirstIntersection(r,scene)
if (no obj) return BackgroundColor;
else begin
  if ( Reflect(obj) ) then
    reflect_color := RayTrace(ReflectRay(r,obj));
  else
    reflect_color := Black;
  if ( Transparent(obj) ) then
    refract_color := RayTrace(RefractRay(r,obj));
  else
    refract_color := Black;
  return Shade(reflect_color,refract_color,obj);
end;
```

Wolfgang Heidrich



## Algorithm Termination Criteria

### Termination criteria

- No intersection
- Reach maximal depth
  - *Number of bounces*
- Contribution of secondary ray attenuated below threshold
  - *Each reflection/refraction attenuates ray*

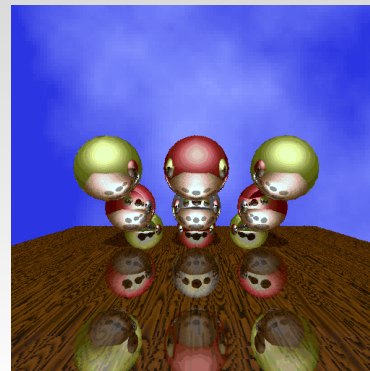
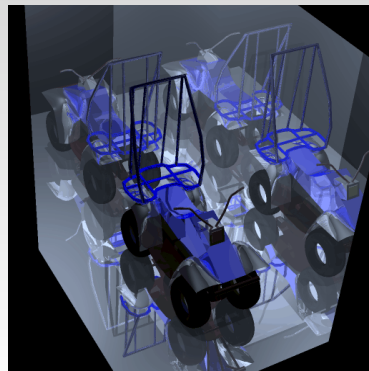
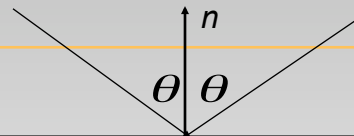
Wolfgang Heidrich



## Reflection

### Mirror effects

- Perfect specular reflection



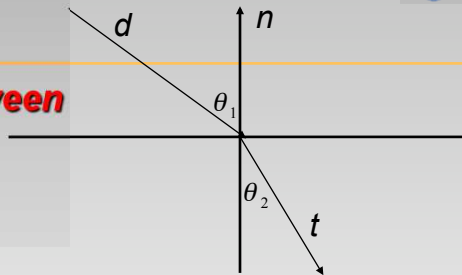
Wolfgang Heidrich



# Refraction

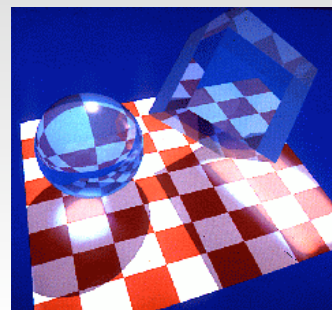
**Happens at interface between transparent object and surrounding medium**

- E.g. glass/air boundary



## Snell's Law

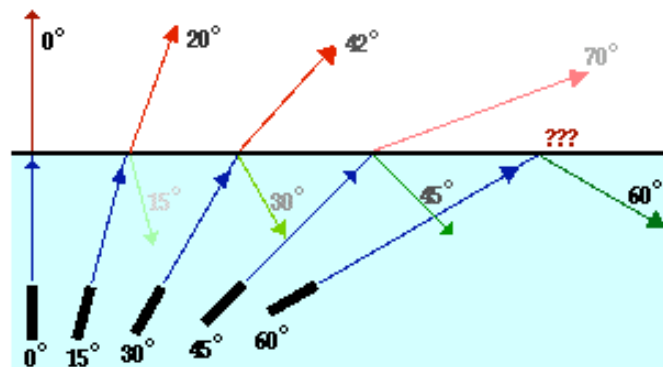
- $c_1 \sin \theta_1 = c_2 \sin \theta_2$
- Light ray bends based on refractive indices  $c_1, c_2$



Wolfgang Heidrich

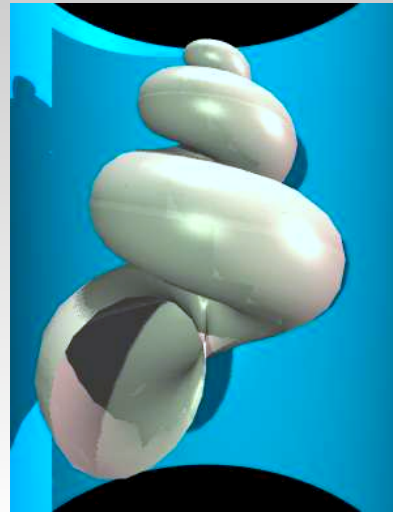
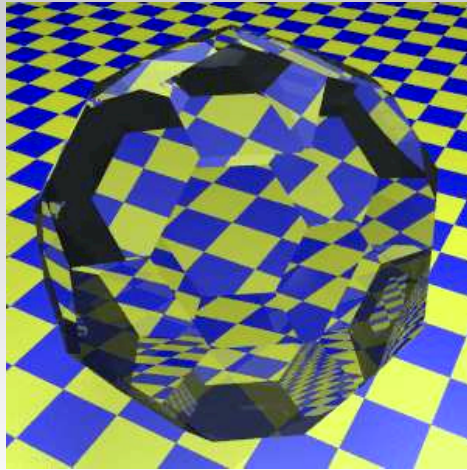
# Total Internal Reflection

As the angle of incidence increases from 0 to greater angles ...



- ...the refracted ray becomes dimmer (there is less refraction)
- ...the reflected ray becomes brighter (there is more reflection)
- ...the angle of refraction approaches 90 degrees until finally a refracted ray can no longer be seen.

## Ray-Tracing Example Images



Wolfgang Heidrich

## Ray-Tracing Terminology



### **Terminology:**

- Primary ray: ray starting at camera
- Shadow ray
- Reflected/refracted ray
- Ray tree: all rays directly or indirectly spawned off by a single primary ray

### **Note:**

- Need to limit maximum depth of ray tree to ensure termination of ray-tracing process!

Wolfgang Heidrich



## Ray-Tracing

### **Issues:**

- Generation of rays
- Intersection of rays with geometric primitives
- Geometric transformations
- Lighting and shading
- **Efficient data structures so we don't have to test intersection with every object**

Wolfgang Heidrich



## Ray Tracing

### **Data Structures**

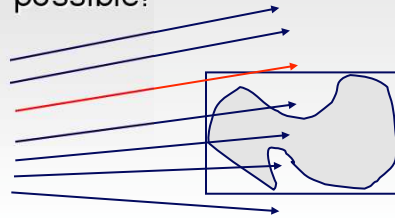
- Goal: reduce number of intersection tests per ray
- Lots of different approaches:
  - *(Hierarchical) bounding volumes*
  - *Hierarchical space subdivision*
    - Oct-tree, k-D tree, BSP tree

Wolfgang Heidrich

## Bounding Volumes

### Idea:

- Rather than testing every ray against a potentially very complex object (e.g. triangle mesh), do a quick *conservative* test first which eliminates most rays
  - Surround complex object by simple, easy to test geometry (typically sphere or axis-aligned box)
    - Want to make bounding volume as tight as possible!

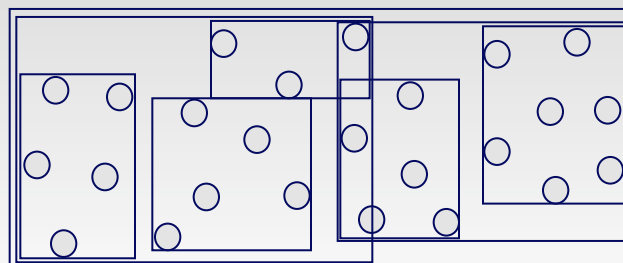


Wolfgang Heidrich

## Hierarchical Bounding Volumes

### Extension of previous idea:

- Use bounding volumes for groups of objects



Wolfgang Heidrich

# Spatial Subdivision Data Structures



## **Bounding Volumes:**

- Find simple object completely enclosing complicated objects
  - Boxes, spheres
- Hierarchically combine into larger bounding volumes

## **Spatial subdivision data structure:**

- Partition the whole space into cells
  - Grids, oct-trees, (BSP trees)
- Simplifies and accelerates traversal
- Performance less dependent on order in which objects are inserted

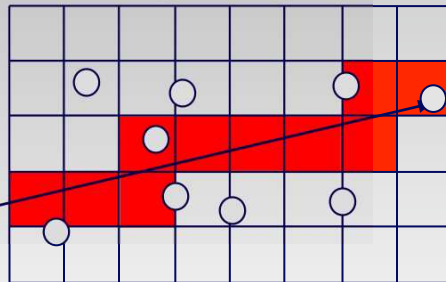
Wolfgang Heidrich

# Regular Grid

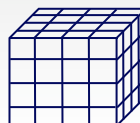


## **Subdivide space into rectangular grid:**

- Associate every object with the cell(s) that it overlaps with
- Find intersection: traverse grid



In 3D: regular grid of cubes (**voxels**):



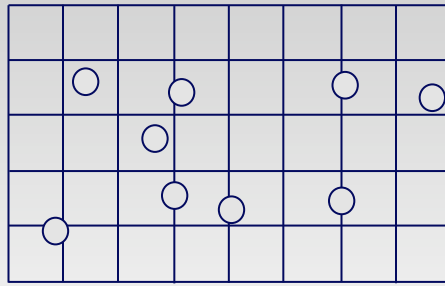
Wolfgang Heidrich



## Creating a Regular Grid

### Steps:

- Find bounding box of scene
- Choose grid resolution in x, y, z
- Insert objects
- Objects that overlap multiple cells get referenced by all cells they overlap



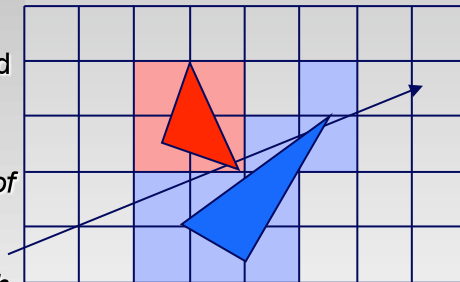
Wolfgang Heidrich



## Grid Traversal

### Traversal:

- Start at ray origin
- While no intersection found
  - Go to next grid cell along ray
  - Compute intersection of ray with all objects in the cell
  - Determine closest such intersection
  - **Check if that intersection is inside the cell**
  - If so, terminate search



Wolfgang Heidrich



## Traversal

### **Note:**

- This algorithm calls for computing the intersection points multiple times (once per grid cell)
- In practice: store intersections for a (ray, object) pair once computed, reuse for future cells

Wolfgang Heidrich



## Regular Grid Discussion

### **Advantages?**

- Easy to construct
- Easy to traverse

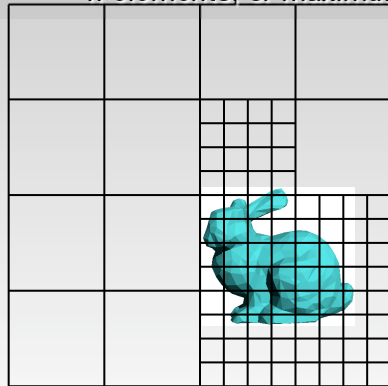
### **Disadvantages?**

- May be only sparsely filled
- Geometry may still be clumped

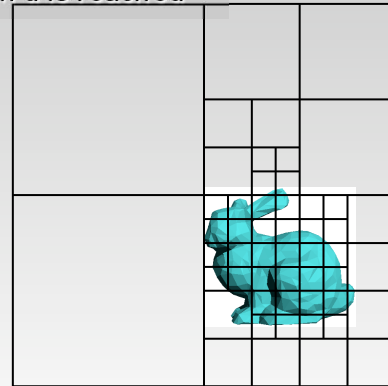
Wolfgang Heidrich

## Adaptive Grids

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



Nested Grids

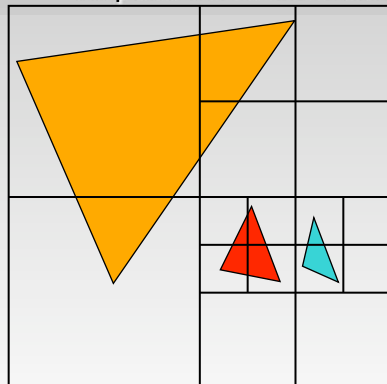


Octree/(Quadtree)

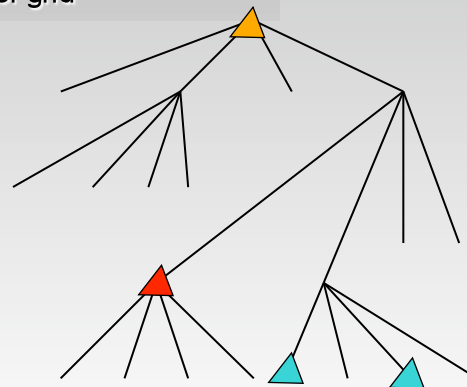
- This slide and the next are curtesy of Fredo Durand at MIT Wolfgang Heidrich

## Primitives in an Adaptive Grid

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



Octree/(Quadtree)





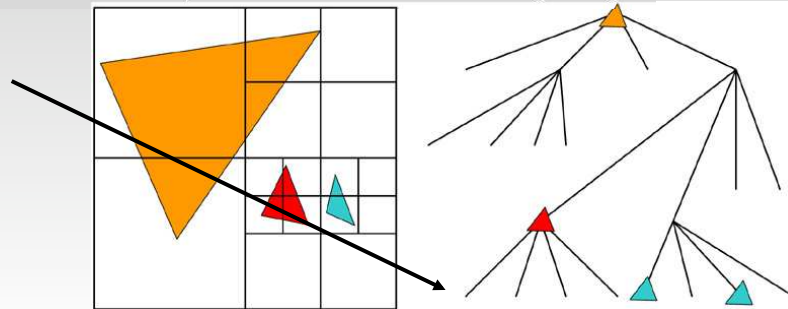
## Adaptive Grid Discussion

### Advantages

- Grid complexity matches geometric density

### Disadvantages

- More expensive to traverse than regular grid



Wolfgang Heidrich

## Coming Up...

### Wednesday/Friday:

- Global illumination

Wolfgang Heidrich