CMT107 Visual Computing

VI.1 Texture Mapping

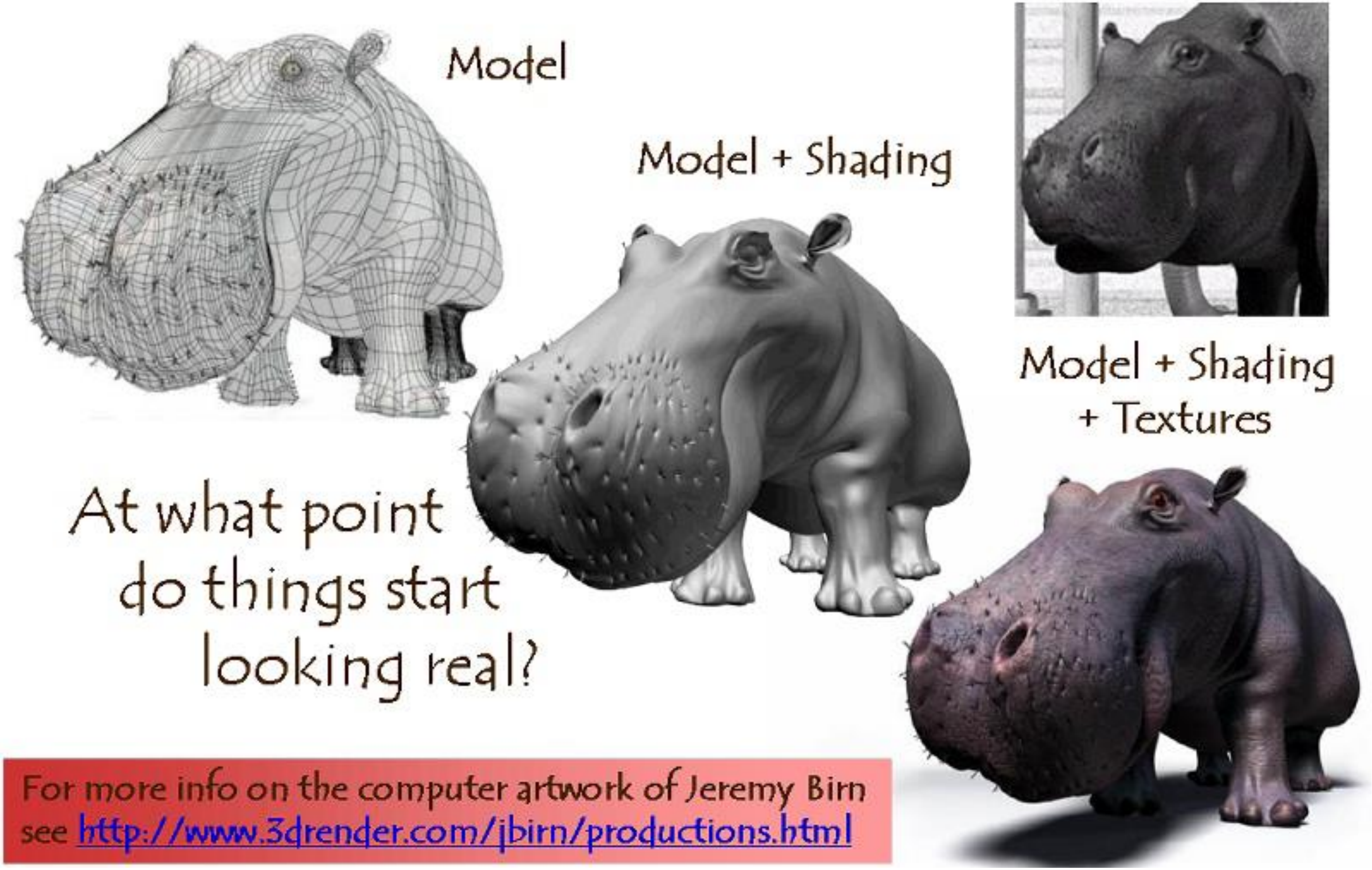
Xianfang Sun

School of Computer Science & Informatics
Cardiff University

Overview

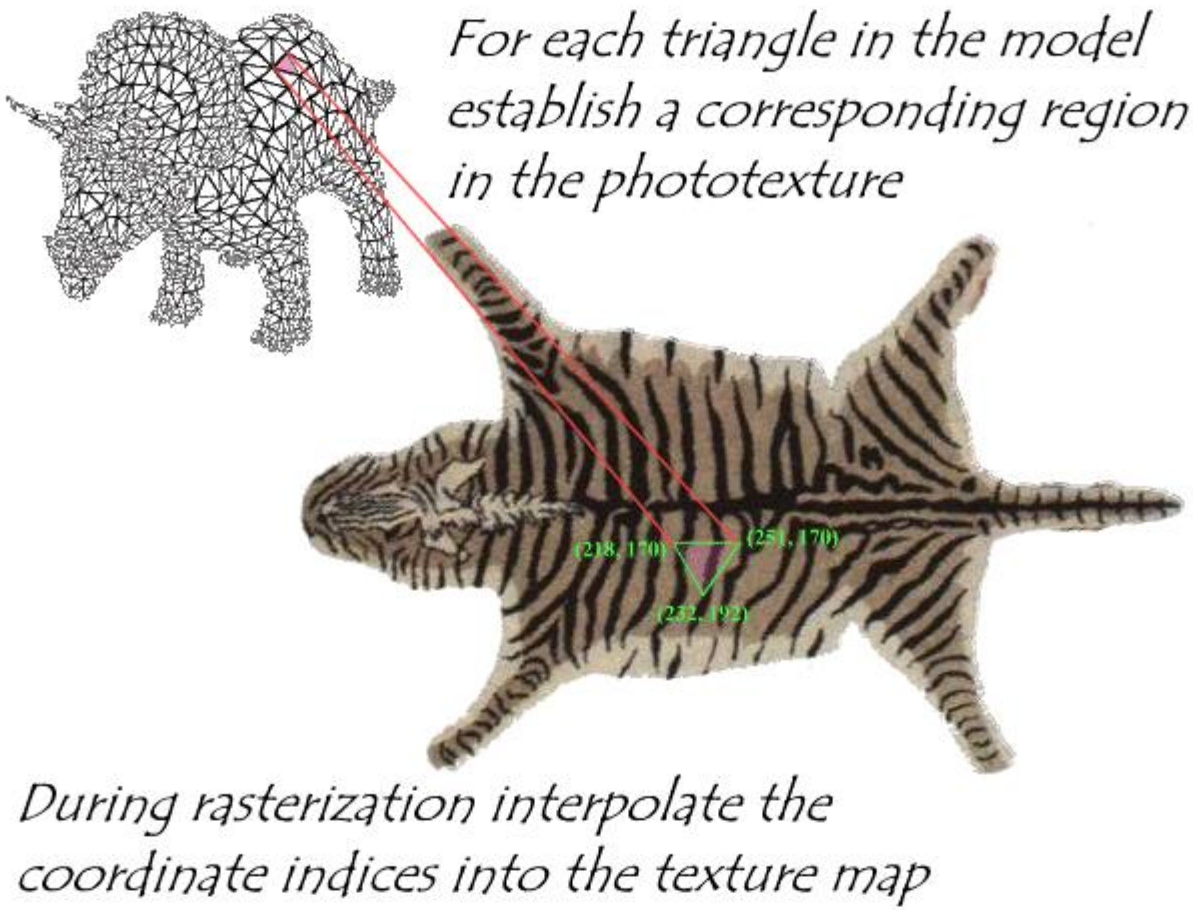
- Texture mapping
  - Texture coordinates
  - Aliasing effects and MIP mapping
- Bump mapping
- Displacement mapping
- Light maps
- Shadow maps
- Texture Mapping in OpenGL

From Shading to Texture



Texture

- Visual appearance of objects can be enhanced by textures
- The concept is simple



Texture Coordinates

- For each vertex specify *texture coordinates* $(s,t) \in [0,1]^2$
  - Canonical position of pixel in texture for vertex
  - For each point p on the 3D polygon, corresponding texture coordinates (s,t) are required

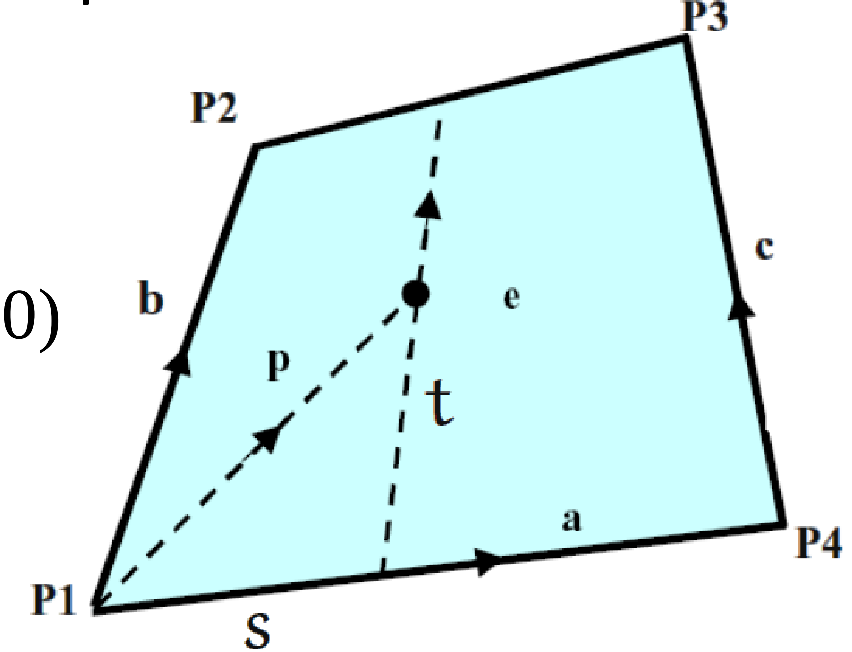
  → *Bilinearly interpolate* texture coordinates in 3D
- Texture coordinates for point on quad

  $p = sa + te$

  $e = b + s(c - b)$

  $p = sa + tb + st(c - b)$

  → Solve for (s,t) (assuming $(0,0)$ is texture coordinate of P_1)



Alias Effects

- One major problem of texture: *alias effects*
  - Caused by *undersampling*; results in *unreal artefacts*

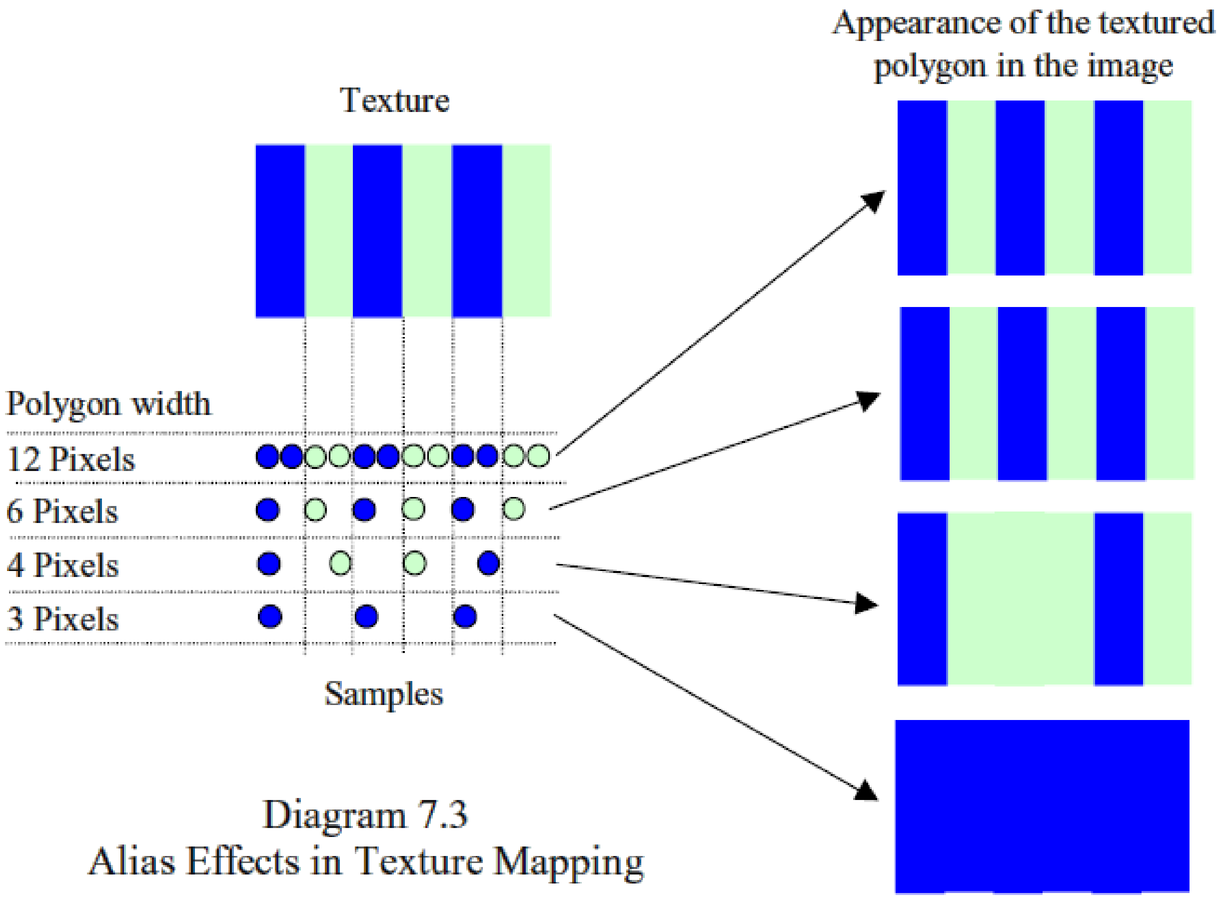


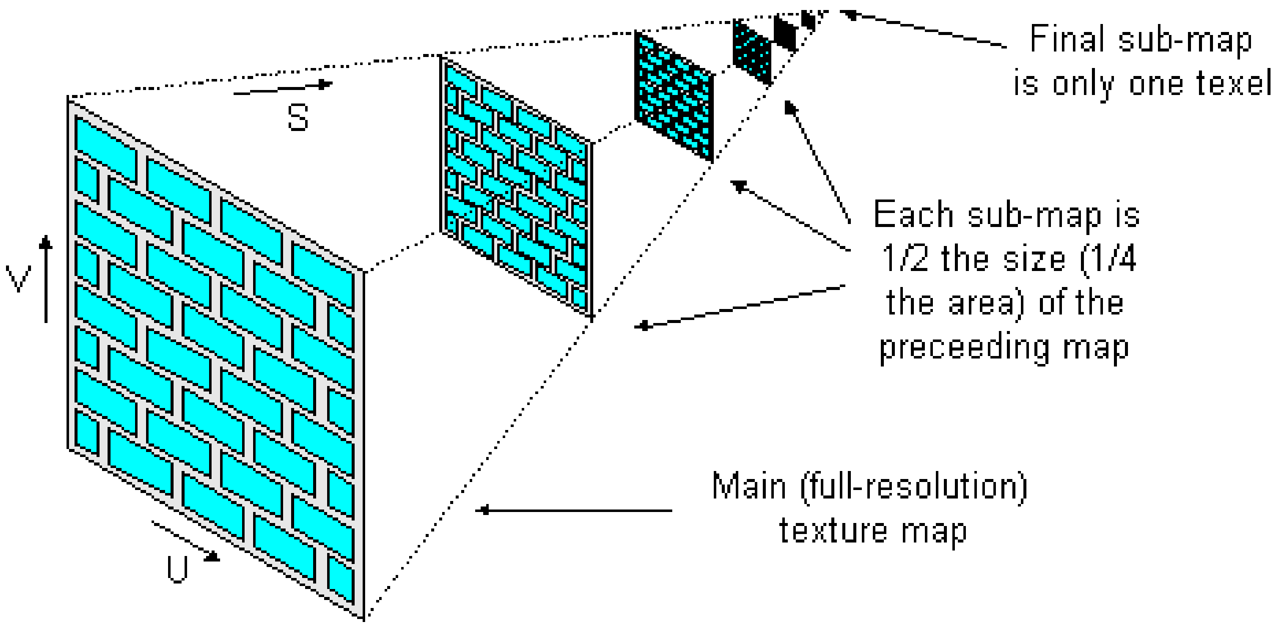
Diagram 7.3
Alias Effects in Texture Mapping

Anti-aliasing

- Similar to untextured images use *anti-aliasing* technique
- Most successful approach: *supersampling*
  - Compute picture at a *higher resolution*
  - *Average* the supersamples to find pixel colour
  - This blurs boundaries, but leaves coherent areas of colour unchanged
  - Works well for polygons, but requires a lot of computations and does not work for line drawings
- Other approaches: convolution filtering (see image processing)

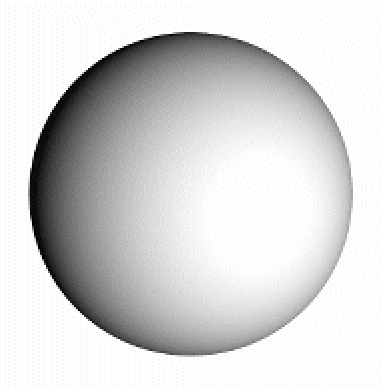
MIP Mapping

- Popular technique of precomputing / prefiltering to address alias effects (MIP = multum in parvo; much in little)
- Basic idea: construct a *pyramid of images* for *different* texture sizes (prefiltered and resampled)
  - Pick texture image suitable for size or interpolate between texture images

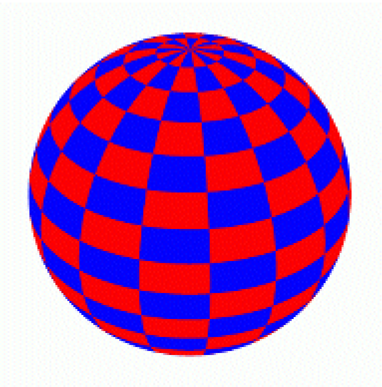


Generalising Texture Mapping

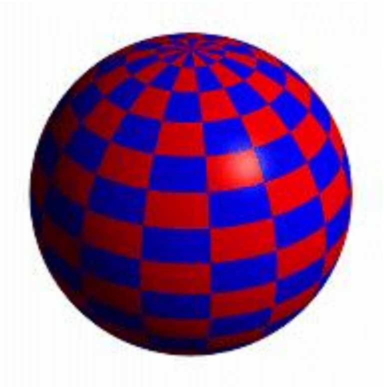
- So far: texture is a *label* (colour) for each pixel
- Can use it to modify other things
  - E.g. use it for *illumination* to adjust material properties (all light types or only some of them)



Material



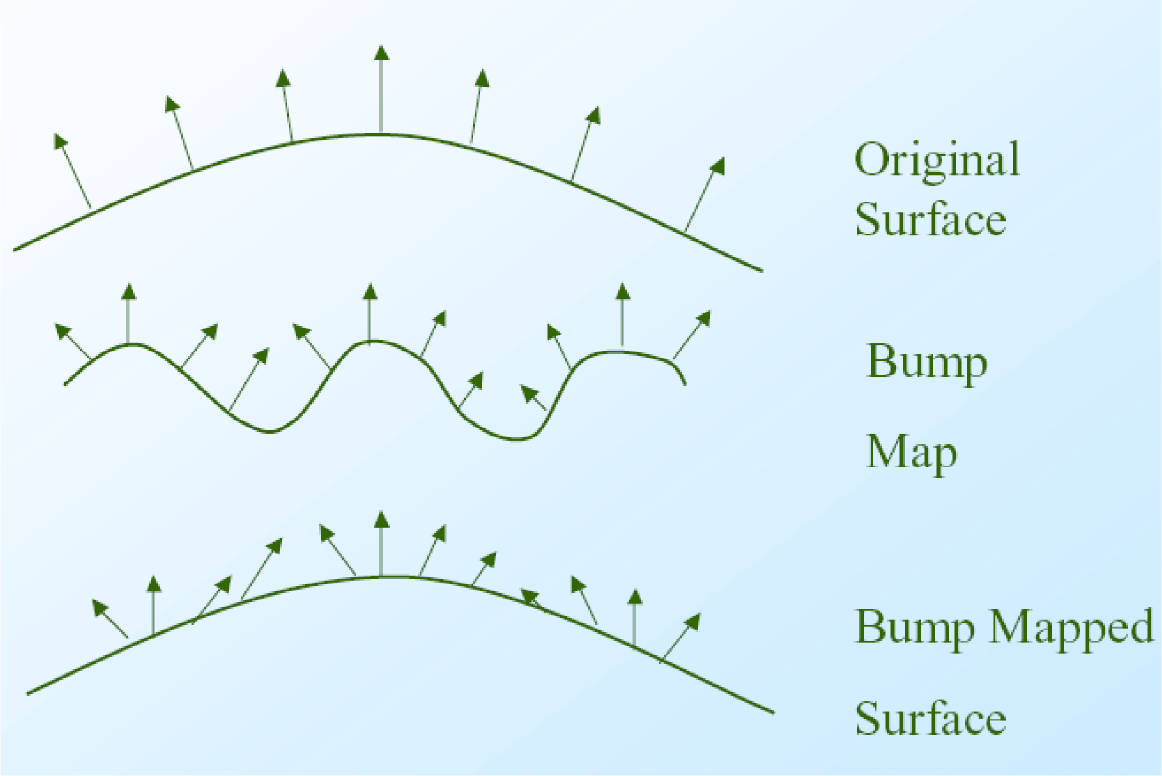
Texture as label



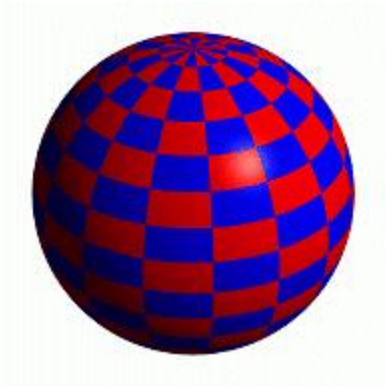
Texture as material

Bump Mapping

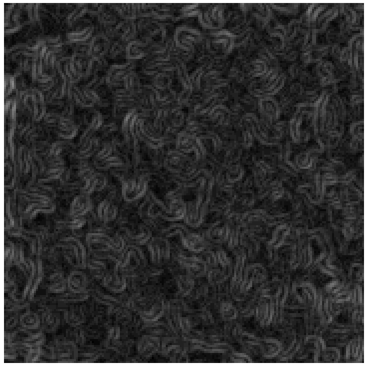
- Texture can be used to alter *surface normals* of an object
  - Does not change shape, but illumination computation
  - Changes in texture (partial derivatives) tell how to change the "height" of the normals



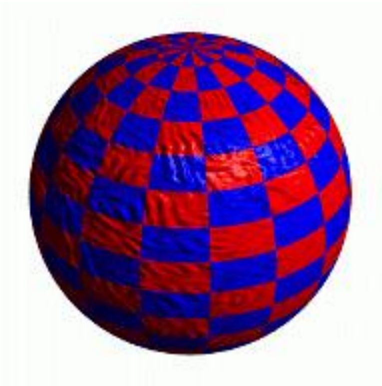
Bump Map Example



Sphere w/Texture



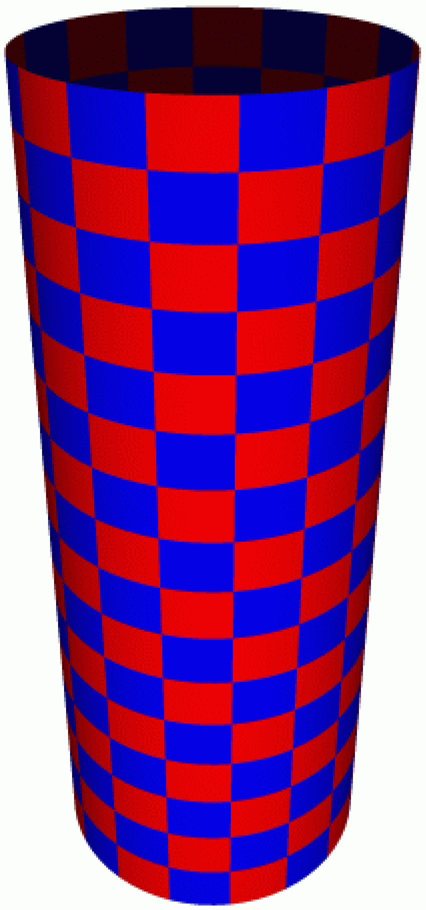
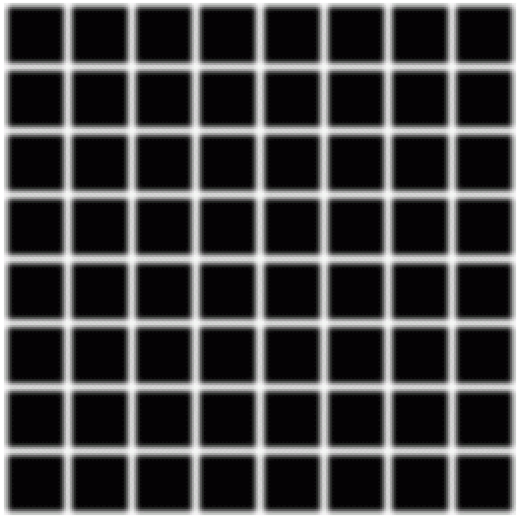
Bump Map



Bumpy Sphere

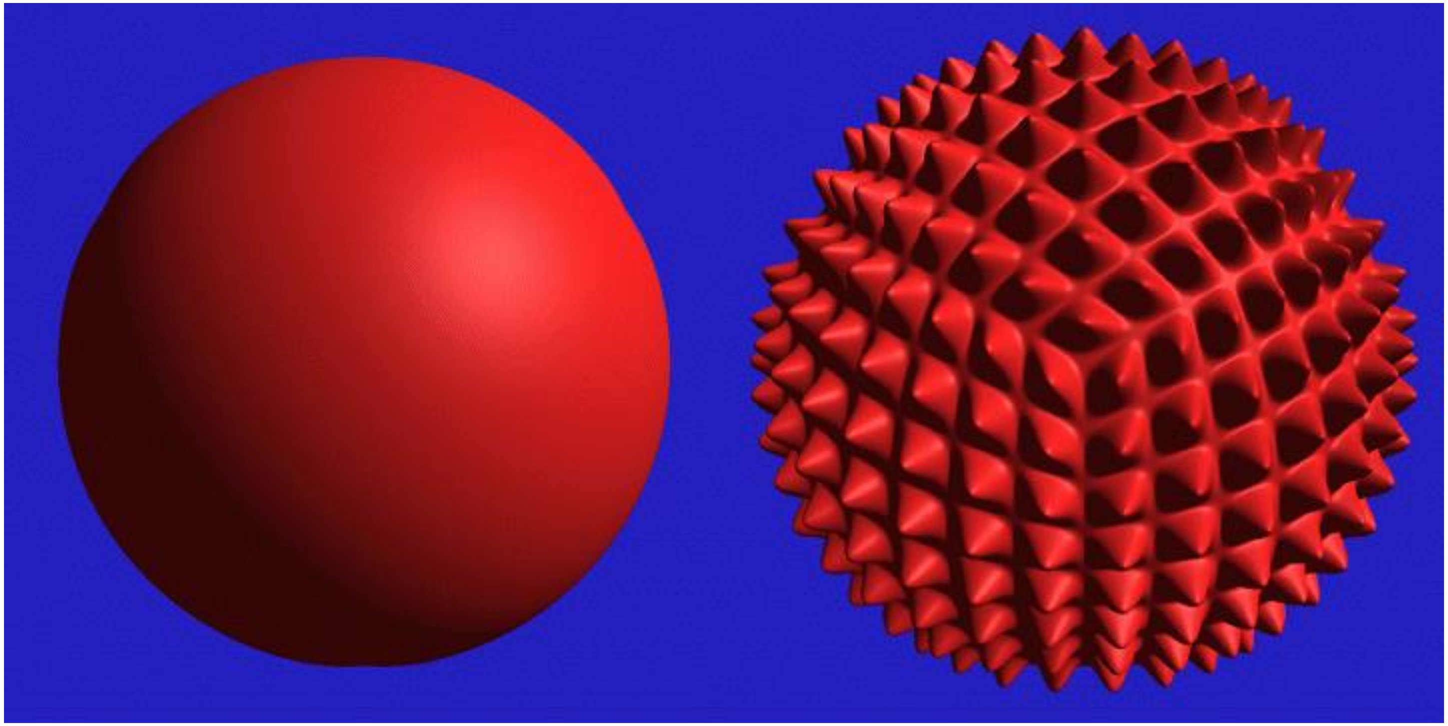
- As we do not change the shape, the silhouette does not change
  - Use only for small bumps
  - Requires illumination computation for each pixel (Phong shading, ray tracing, . . .)

Another Bump Map Example


Displacement Mapping

- Use texture map to *move* surface points



Light Maps

- In Quake *texture* and *light maps* are used
  - Light map contains precomputed illumination at low resolution
  - Multiply light map with texture map at run-time (and cache it)

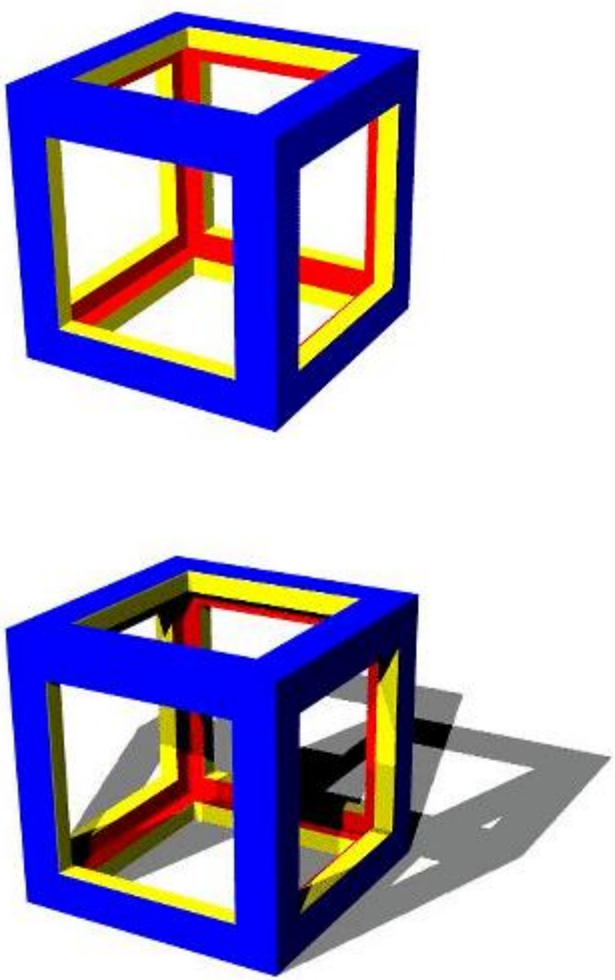
Only Texture Map

Texture and Light Map

Shadow Maps

- Generate *shadows* using texture maps
  - Render scene from the *viewpoint of each light source* and only keep *depth buffer* values in *shadow buffers*
  - When shading each pixel (illumination computation per pixel):
    - Compute vector *L* from visible point to light source (needed for illumination computation)
    - Compute the length of *L*
    - *Compare* this length with corresponding value in the *shadow buffers*
    - If the shadow buffer value is *less*, then the point is in the shadow and we can ignore the light source

Shadow Map Example



Texture Mapping in OpenGL

- Use Texture and TextureIO to apply a texture
  1. Create a texture object using TextureIO
     - TextureIO.newTexture(File, boolean);
  2. Indicate how the texture is to be applied to each pixel
     - Texture.setTexParameteri (…)
  3. Draw the scene, supplying both texture and geometric coordinates; send the coordinates to vertex shader, and send texture sampler to fragment shader
     - Texture.getImageTexCoords().top() …

Texture Mapping in OpenGL

- Using OpenGL Core functions to apply a texture
  1. Create a texture object and specify a texture for that object
     - glGenTextures(…)
     - glBindTexture(…)
     - glTexImage2D(…)
  2. Indicate how the texture is to be applied to each pixel
     - glTexParameteri(…)
  3. Enable texture mapping
     - glEnable(GL_TEXTURE_2D)
  4. Draw the scene, supplying both texture and geometric coordinates; send the coordinates to vertex shader, and send texture sampler to fragment shader

- Step 0: Read in texture image

Texture Object

- Texture objects store texture data and keep it readily available for usage. Many texture objects can be generated.
- Generate identifiers for texture objects first

  int texids[n];

  glGenTextures(n, texids)
  - n: the number of texture objects identifiers to generate
  - texids: an array of unsigned integers to hold texture object identifiers
- Bind a texture object as the current texture

  glBindTexture(target, identifier)
  - target: can be GL_TEXTURE_1D, GL_TEXTURE_2D, or GL_TEXTURE 3D
  - identifier: a texture object identifier
- Specify texture image

  glTexImage2D(target, level, internalFormat,
  width, height, border, format, type, data);

Texture Object Example Code

```
int texids[] = new int[1];
ByteBuffer texImg = readImage("textures/Day.png");

glGenTextures(1, texids, 0);
glBindTexture(GL_TEXTURE_2D, texids[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             texWidth, texHeight,0, GL_BGR,
             GL_UNSIGNED_BYTE, texImg);
```

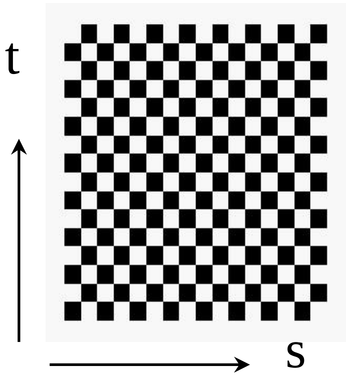
Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied.
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Environment parameters determine how texture mapping interacts with shading
  - Mipmapping allows us to use textures at multiple resolutions
- OpenGL Command

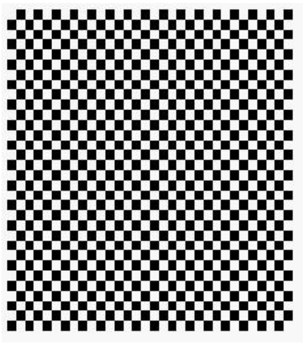
  glTexParameterf(target, pname, param);
  - target: Specifies the target texture
  - pname: Specifies the symbolic name of a single-valued texture parameter
  - param: Specifies the value of pname.

Wrapping Modes

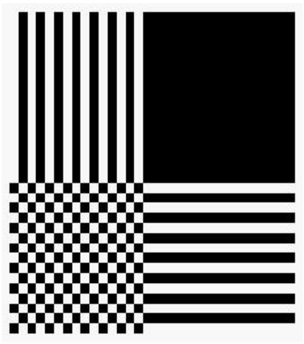
- Repeat: use s,t modulo 1
- Clamp: if s,t > 1 use 1, if s,t <0 use 0
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)
  - GL_CLAMP_TO_BORDER, GL_MIRRORED_REPEAT…



texture



GL_REPEAT wrapping



GL_CLAMP_TO_EDGE wrapping

Texture Filtering

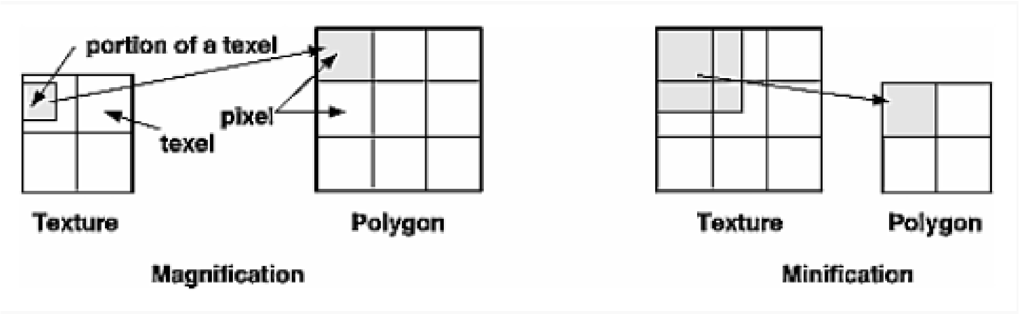
- A pixel may be mapped to a small portion of a texel or a collection of texels from the texture map. How to determine the color of the pixel?
- Magnification: when a pixel mapped to a small portion of a texel

  glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MAG_FILTER, type);

  - type: GL_NEAREST or GL_LINEAR
- Minification: when a pixel mapped to many texels

  glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER, type);

  - type: GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR, …



Shading and Texture Interaction

- You can specify how the texture-map colors are used to modify the pixel colors by setting environment parameters in old version of OpenGL

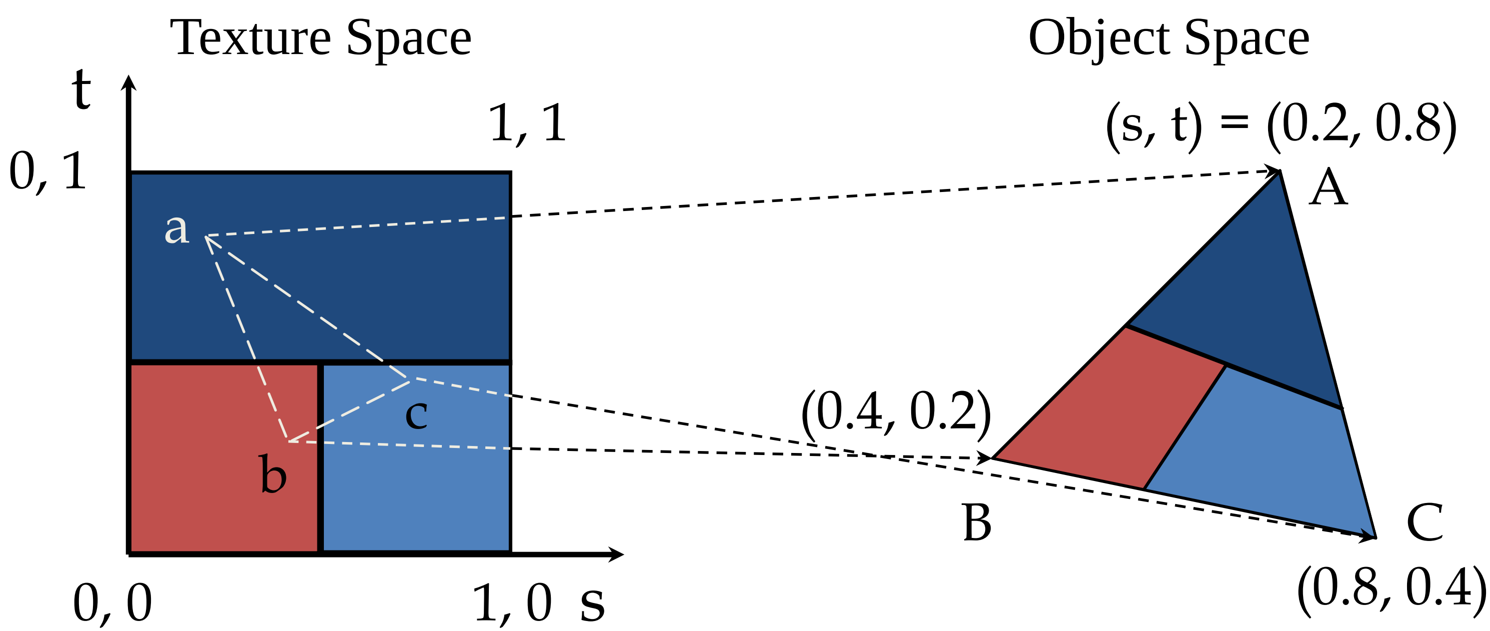
  glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, mode);

  mode values:

  - GL_REPLACE : replace pixel color with texture color
  - GL_BLEND : $C = C_f(1\text{-}C_t) + C_cC_t$,
    - C_f is the pixel color, C_t is the texture color, and C_c is some constant color
  - GL_MODULATE : $C = C_fC_t$ (Default)
  - More on OpenGL programming guide
- In the shader version of OpenGL, the interaction should be implemented in the fragment shader.

Assign Texture Coordinates

- Every point on a surface should have a texture coordinate (s, t) in texture mapping
- We often specify texture coordinates to polygon vertices and interpolate texture coordinates with the polygon
- Texture.getImageTexCoords() can be used to retrieve texture coordinates



Typical Code in Main Program

```
// Set the texture to be used
try {
      texture = TextureIO.newTexture(new File("WelshDragon.jpg"), false);
    }  catch (IOException ex) {
        Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
    }

    // Set texture coordinates
    float[] texCoord ={…};
    FloatBuffer textures = FloatBuffer.wrap(texCoord); gl.glGenBuffers(…);
    gl.glBindBuffer(…);
    gl.glBufferData(…);
    gl.glBufferSubData(…);

    // Send texture coordinates to vertex shader
    vTexCoord = gl.glGetAttribLocation( program, "vTexCoord" );
    gl.glEnableVertexAttribArray( vTexCoord );
    gl.glVertexAttribPointer( vTexCoord, 2, GL_FLOAT, false, 0, offsetSize);

    // Set the fragment shader texture sampler variable
    gl.glUniform1i( gl.glGetUniformLocation(program, "tex"), 0 );
```

Vertex Shader

```
#version 330 core

layout(location = 0) in vec4 vPosition;
layout(location = 1) in vec3 vColour;
layout(location = 2) in vec2 vTexCoord;

out vec4 color;
out vec2 texCoord;

uniform mat4 ModelView;
uniform mat4 Projection;

void main()
{
    gl_Position = Projection * ModelView * vPosition;
    texCoord    = vTexCoord;

    color.rgb = vColour;
    color.a = 1.0;
}
```

Fragment Shader

```
#version 330 core

in  vec4 color;
in  vec2 texCoord;

out vec4 fColor;

uniform sampler2D tex;

void main()
{
    fColor = color* texture( tex, texCoord );
}
```

More details in the Labs...

Summary

- Describe the principle of texture maps. What are texture coordinates and how are they related to 3D coordinates?
- What options do exist to generalise texture maps? For what other effects are they useful and what are the advantages and disadvantages of these techniques?
- How to program texture mapping in OpenGL?

CMT107 Visual Computing

VI.2 Ray Tracing

Xianfang Sun

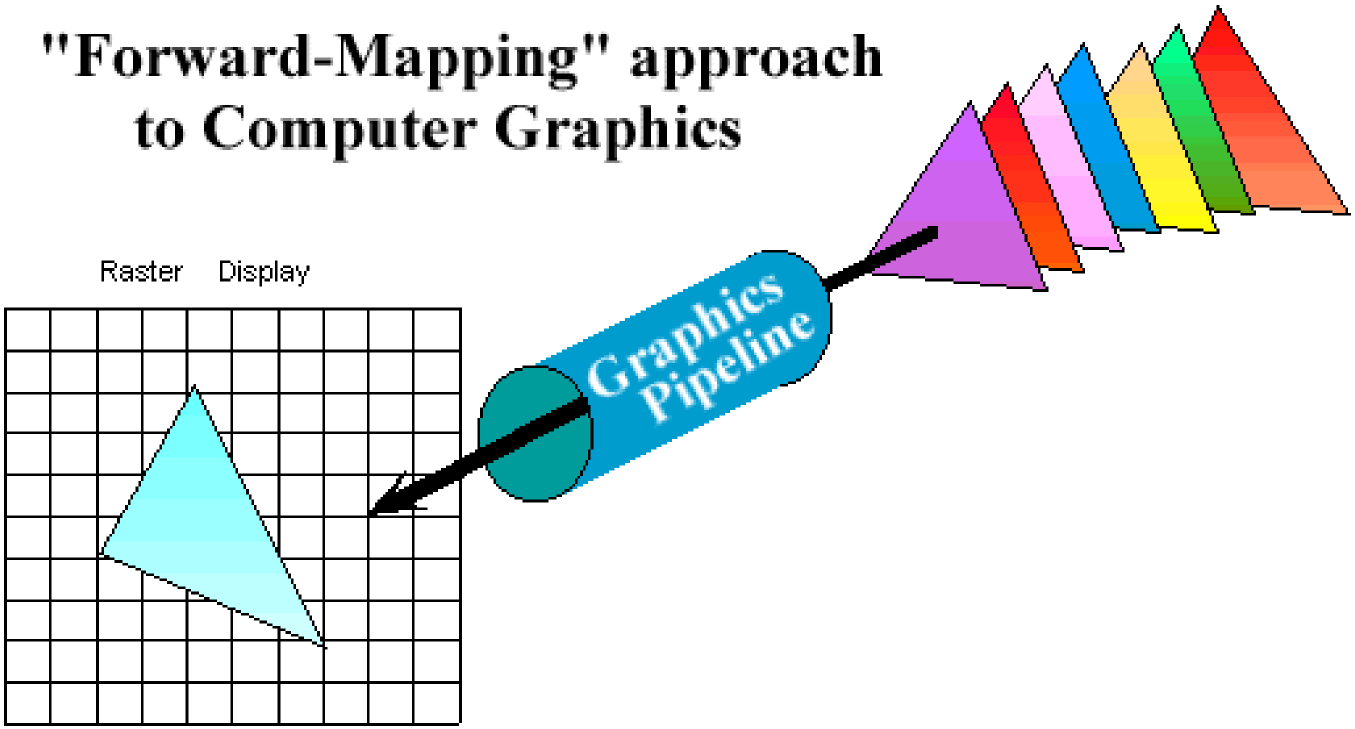
School of Computer Science & Informatics
Cardiff University

Overview

- Ray casting
- Ray tracing

Graphics Pipeline Review

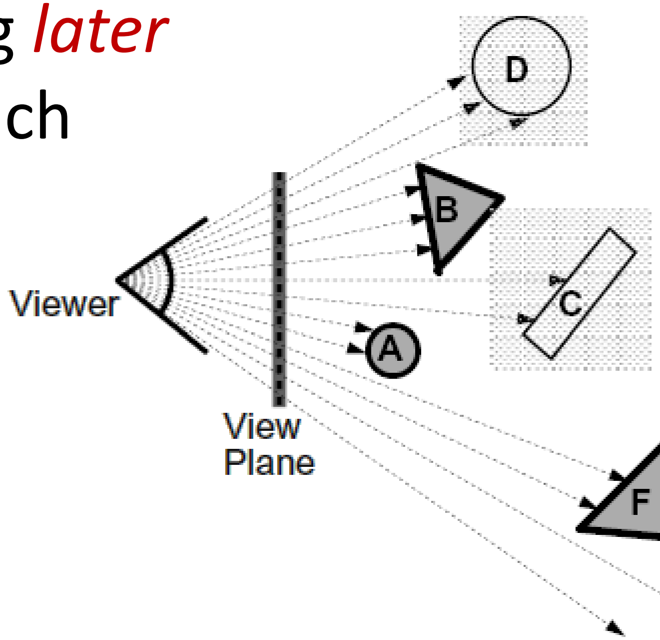
- Properties of the graphics pipeline
  - Primitives are processed *one at a time* (in sequence)
  - All analytic processing done *early on*
  - Scan conversion (Rasterisation) occurs *last*
  - Minimal state required (*immediate mode* rendering)



Ray Casting

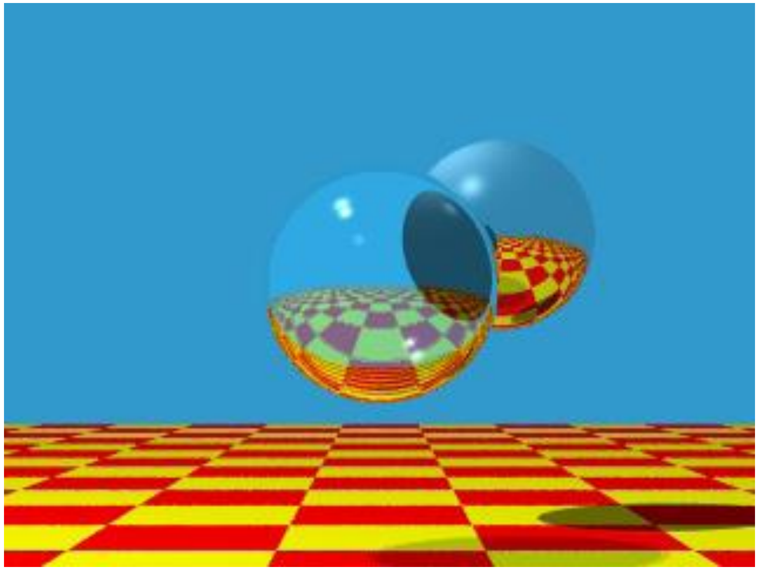
- An alternative to pipeline approach: *ray casting*
  - Search along lines of sight (*rays*) for visible primitive(s)
- Properties:
  - Go through *all primitives* at *each pixel*

    (must have all primitives in a display list)
  - Sample (rasterisation) *first*
  - Do analytic processing *later*
- *Inverse mapping* approach



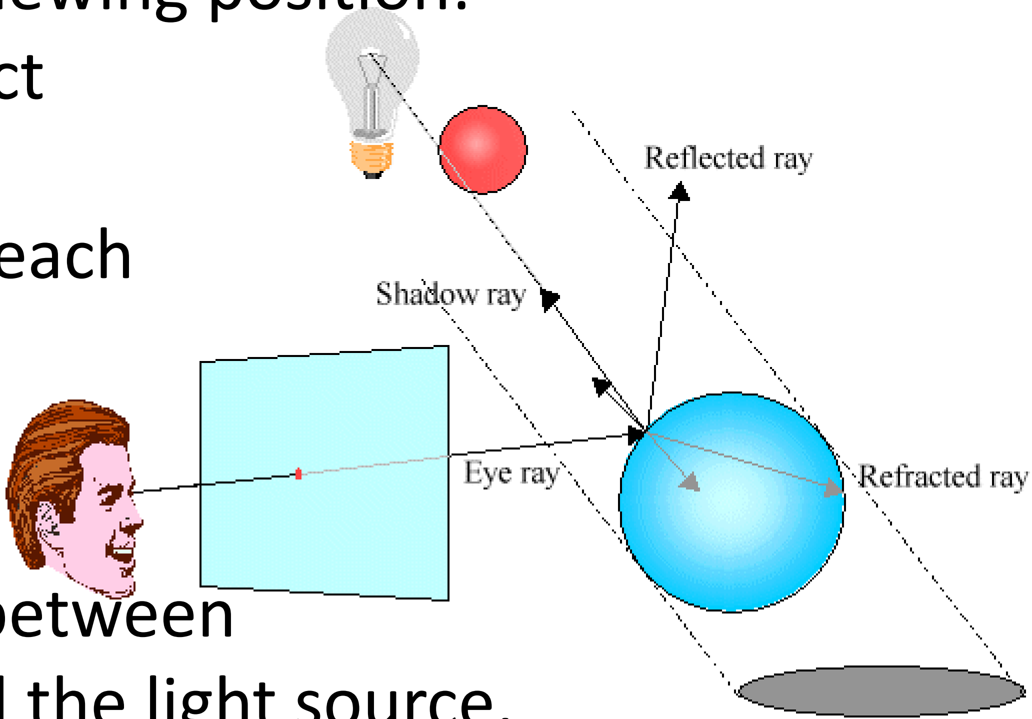
Global Illumination

- Ray casting properties:
  - Takes no advantage of *screen* space coherence
  - Requires *costly visibility computation*
  - Forces *per pixel illumination* evaluations
  - Not suited for immediate mode rendering
- In 1980 T. Whitted introduced *recursive ray casting (ray tracing)* to address global illumination



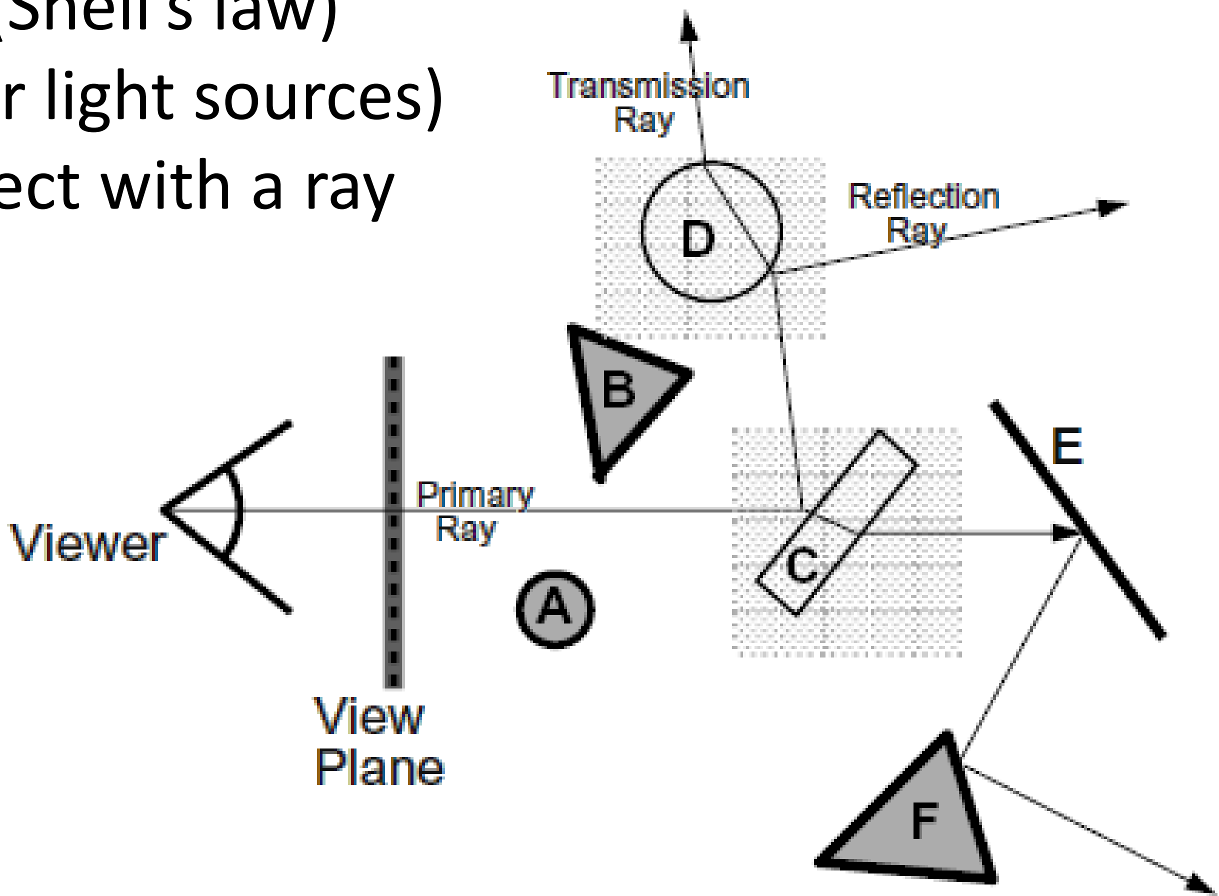
Ray Tracing

- For each ray from the viewing position:
  - Compute *visible* object along the ray
  - Compute *visibility* of each light source from the visible surface point using a new ray
  - If there is an object between the surface point and the light source, *ignore* the light source; otherwise, Phong illumination model is used to evaluate the light intensity
  - Can easily add reflection and refraction, etc.



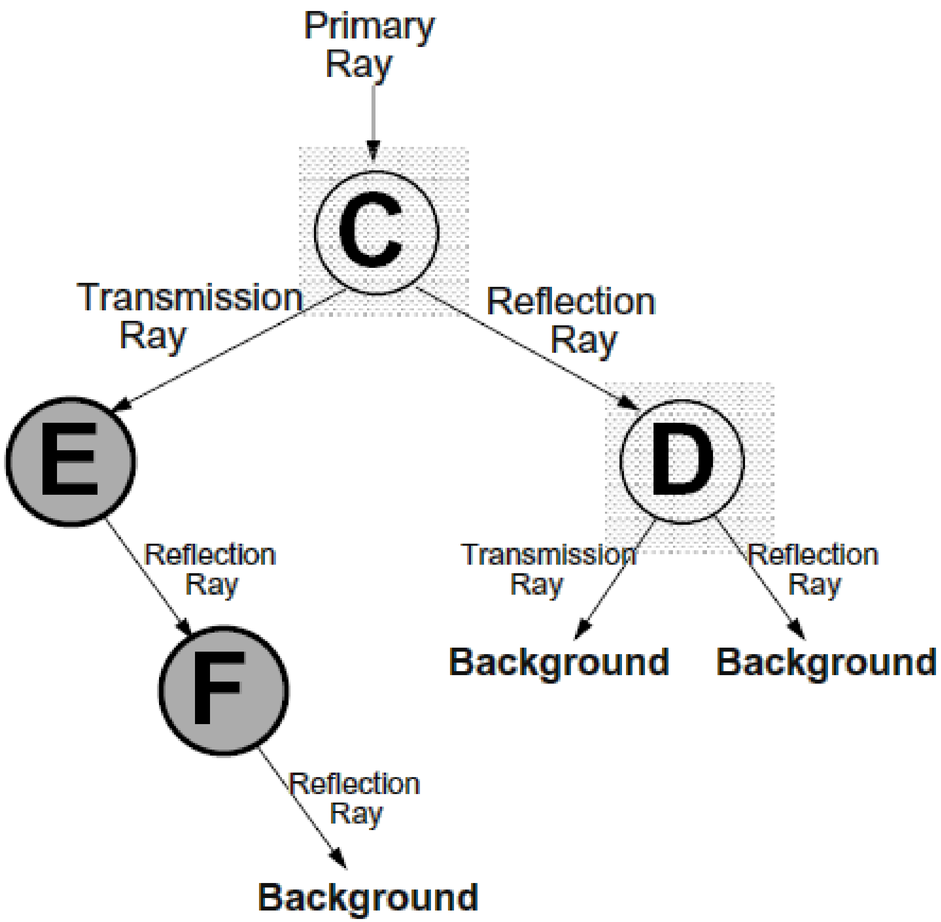
Ray Tracing

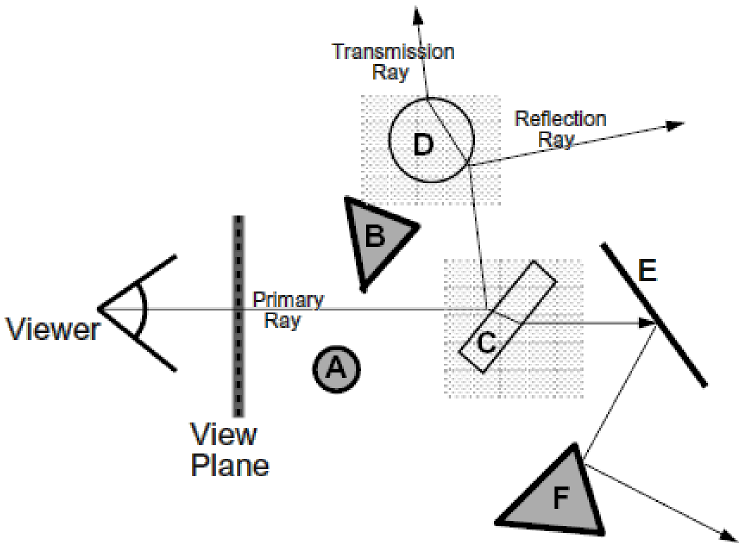
- For each object we need to know how to
  - *reflect* light (Phong's illumination model)
  - *refract* light (Snell's law)
  - *emit* light (for light sources)
  - *intersect* object with a ray



Ray Tracing Tree

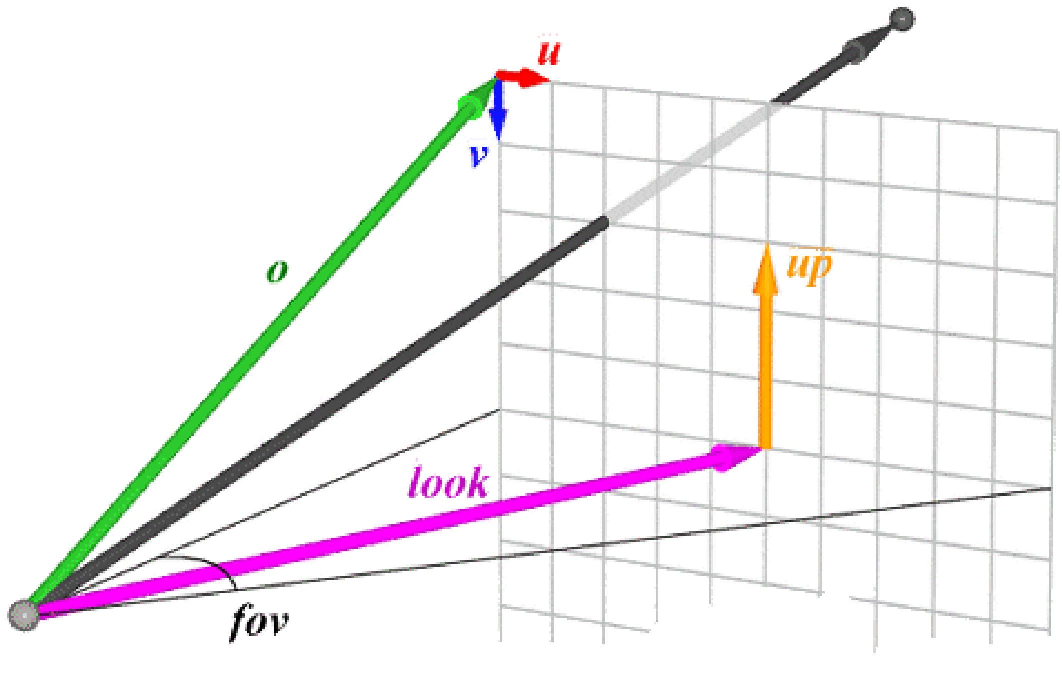
- Move up backwards in tree and combine intensities as determined by *Phong's illumination model*





From Pixels to Rays

- Compute ray direction $v(x, y)$ for raster coordinates (x, y)



- $$u = \frac{\text{look} \times \text{up}}{\|\text{look} \times \text{up}\|}$$

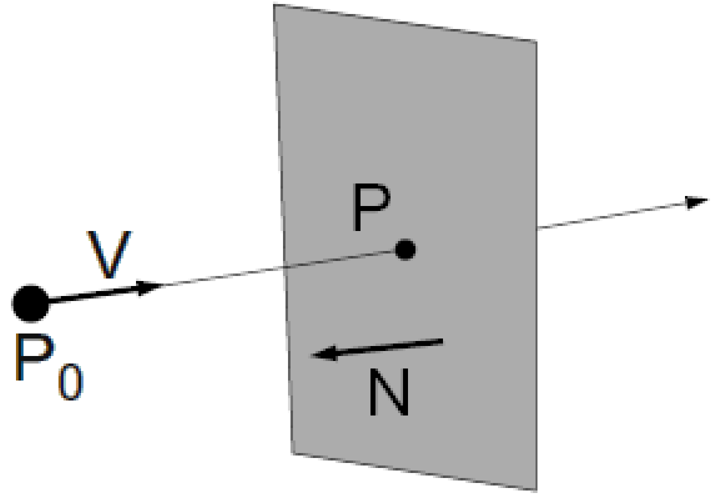
- $$v = \frac{\text{look} \times u}{\|\text{look} \times u\|}$$

- $$o = \frac{\text{look}}{\|\text{look}\|} \frac{\text{width}}{2\tan\left(\frac{\text{fov}}{2}\right)} - \frac{\text{width}}{2} u - \frac{\text{height}}{2} v$$

- $$v(x, y) = (x u_x + y v_x + o_x; x u_y + y v_y + o_y; x u_z + y v_z + o_z)$$

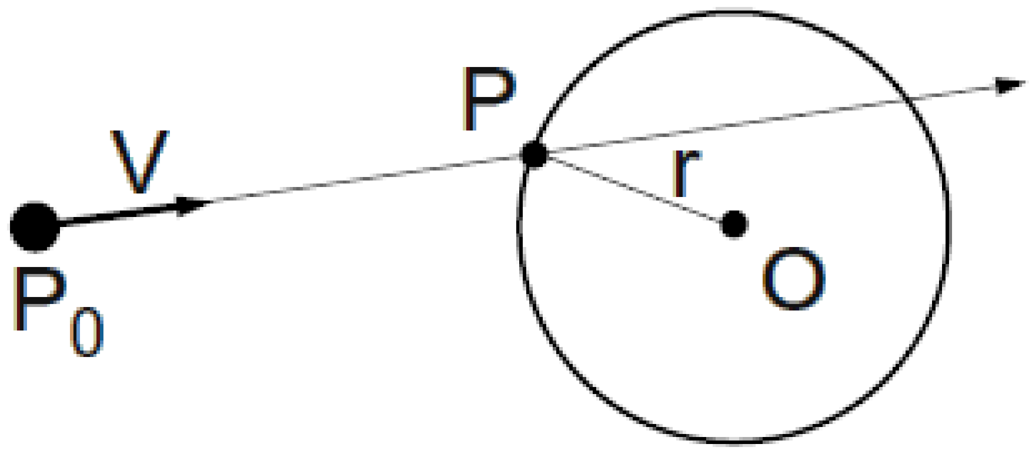
Ray-Plane/Polygon Intersection

- *plane-line intersection*
  - Ray: $P = P_0 + tV$
  - Plane: $P^{T}N + D = 0$
  - Substitute: $(P_0 + tV)^{T}N + D = 0$
  - Solution: $t = -\,(P_0{}^{T}N + D)\,/(V^{T}N)$
- For intersection with polygon, check if intersection point lies inside polygon



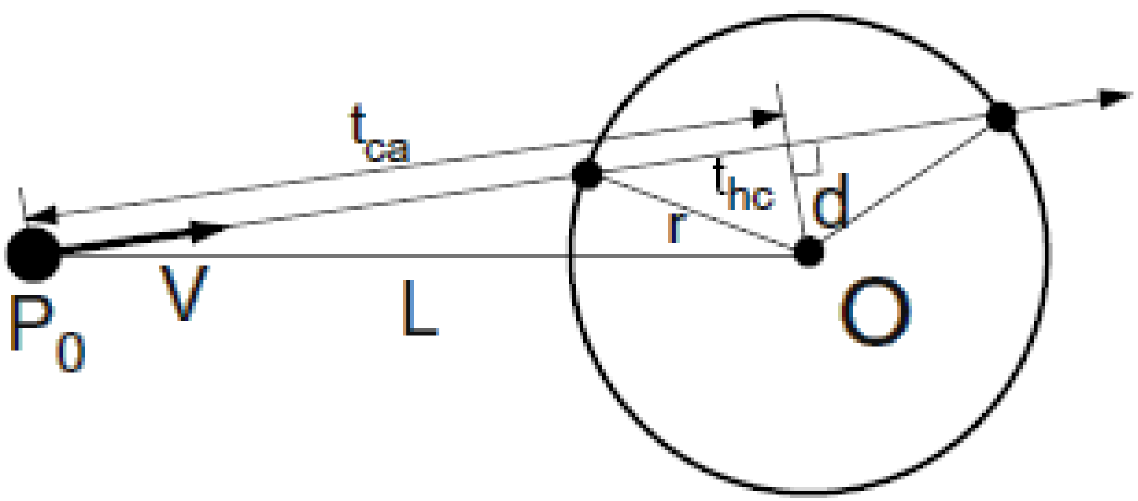
Ray-Sphere Intersection

- *Intersect* a sphere with the ray (*algebraic*)
  - Ray parameterisation: $P(t) = P_0 + tV$
  - Sphere equation: $\|P - O\|^2 - r^2 = 0$
  - Substitute: $\|P_0 + tV - O\|^2 - r^2 = 0$
  - Solve: $t^2 + 2V^T(P_0 - O)t + (\|P_0 - O\|^2 - r^2) = 0$



Ray-Sphere Intersection

- *Intersect* a sphere with the ray (*geometric*)
  - $L = O - P_0$, $t_{ca} = L^T V$
  - if $t_{ca} < 0$, no intersection
  - $d^2 = L^T L - t_{ca}^2$
  - if $d > r$, no intersection
  - $t_{hc} = \sqrt{r^2 - d^2}, \rightarrow \quad t = t_{ca} - t_{hc}$ and $t_{ca} + t_{hc}$



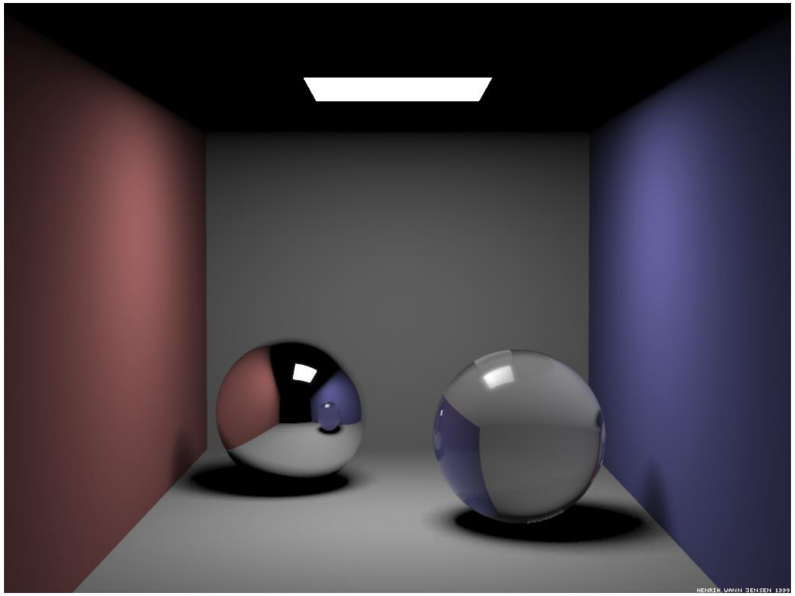
Ray Tracing Summary

- Input: viewing position ***v***, look-at point ***a*,** up vector ***u***
- For each pixel:
  - Create a ray ***l*** from the viewing position ***v*** in direction ***d*** such that it passes the pixel in the viewing plane
  - Set the colour to be the return value of *raytrace(**v**, **d**)*
- Function *raytrace (**v**, **d**)***:**
  - Initialise position ***t*** on ray ***l*** from ***v*** in direction ***d*** to infinity and the nearest object ***n*** to empty
  - For each object ***o*** in the scene
    - Compute intersection ***p*** of ***l*** and ***o*** closest to ***v***
    - If ***p*** exists and is closer to ***v*** than ***t***, set ***t*** to ***p*** and ***n*** to ***o***
  - If ***n*** is empty, return background colour, else ...

Ray Tracing Summary (cont.)

- else …
  - If ***n*** is reflective and we haven't reached the maximum recursion depth level, compute perfect reflection vector ***r*** of ***d*** at ***t*** and call *raytrace**(t, r)*** to obtain reflected colour c_r
  - If ***n*** is transparent and we haven't reached the maximum recursion depth level, compute refraction vector ***r'*** of ***d*** at ***t*** and call *raytrace**(t, r')*** to obtain refracted colour c_t
  - For each light source $k=1, \ldots, m$ at position $\boldsymbol{l}_k$, cast ray from ***t*** to $\boldsymbol{l}_k$. If this line segment intersects with any of the other objects, ***t*** is in the shadow of this object. Otherwise compute the amount of light c_k reaching ***t*** *from k*
  - Return combination of colours c_r, c_t *and* c_k, $k=1, \ldots, m$

Examples

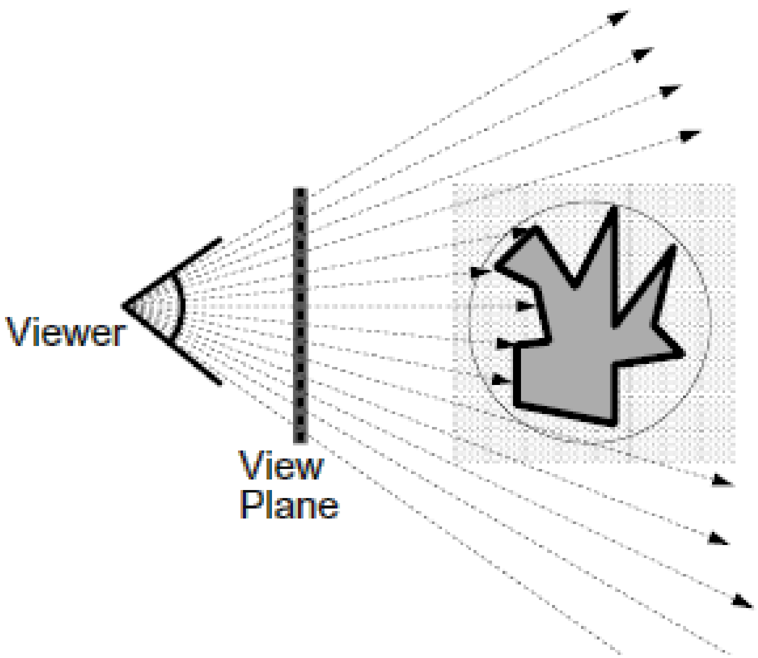


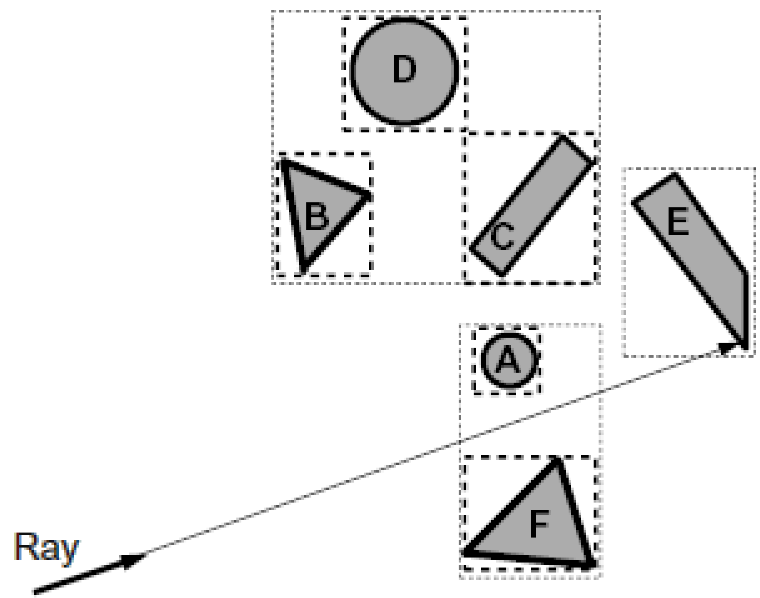
Properties of Ray Tracing

- *Advantages*
  - Improved realism (shadows, reflections, transparency)
  - Higher level rendering primitives
  - Very simple design
- *Disadvantages*
  - Very slow per pixel calculations
  - Only approximate global illumination
    (cannot follow all rays)
  - Hard to accelerate with hardware
- *Acceleration* approach
  - Try to reduce number of intersection computations

Ray Tracing Acceleration

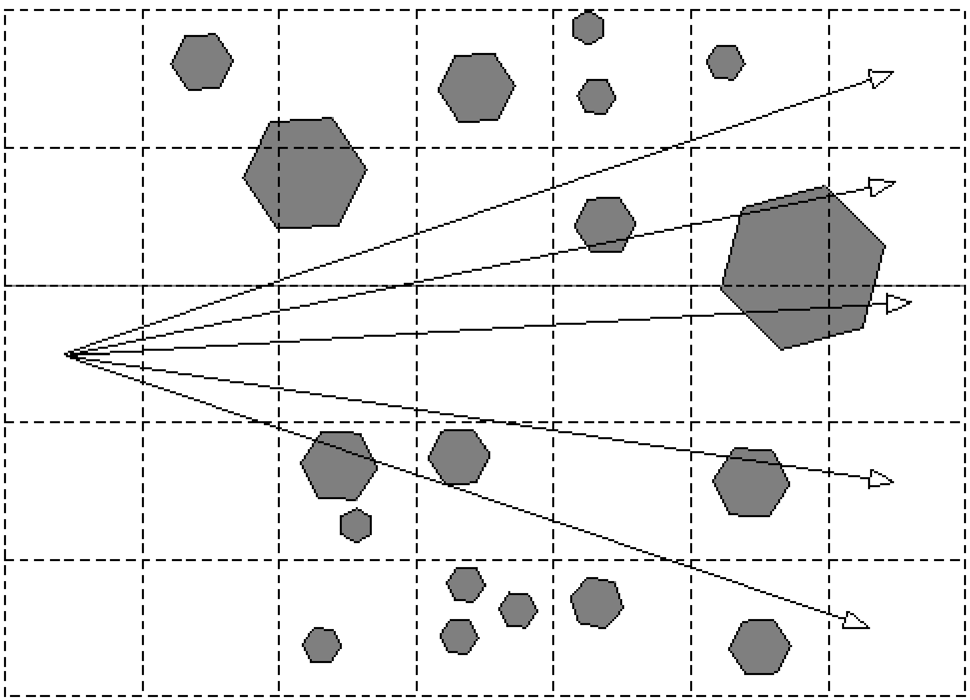
- *Bounding volumes*
  - Check simple bounding volume for ray/surface intersections before checking complex shapes
- *Bounding volume hierarchies*
  - Construct and check hierarchical bounding volumes





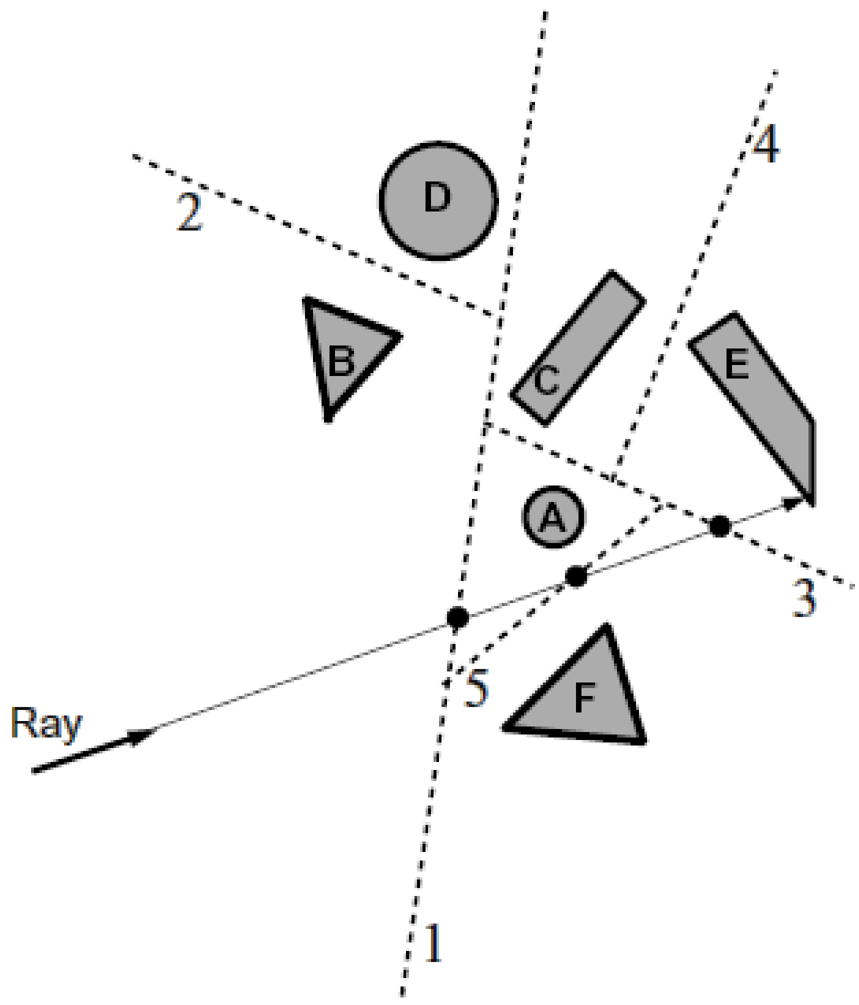
Spatial Data Structures

- Create a data structure aware of the spatial relations
  - *Partition* space and place objects within subregions
  - *Only* consider subregions that the ray passes through
  - *Avoid computing intersections twice* if object lies inside multiple subregions

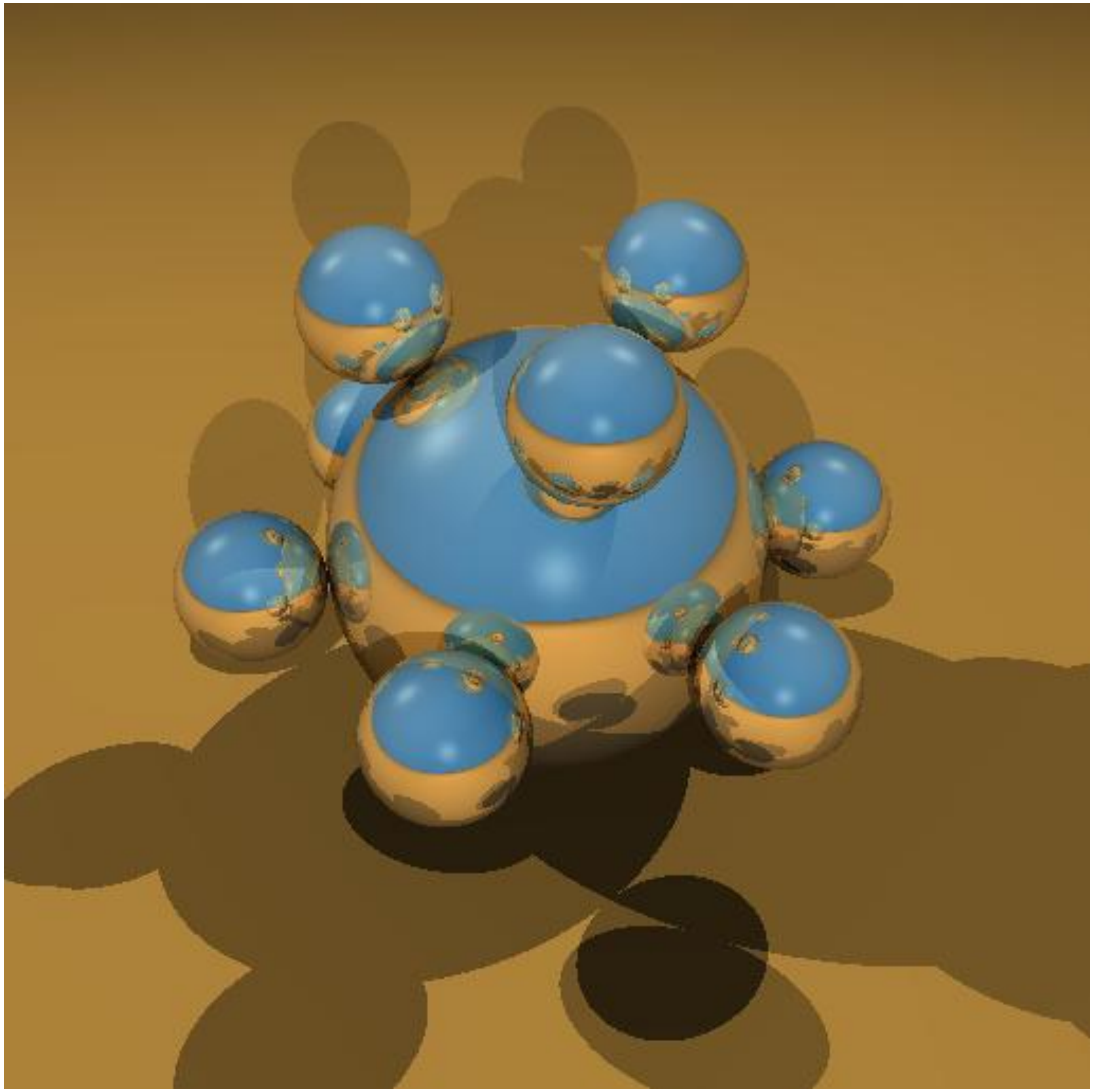


BSP Trees in Ray Tracing

- Partition space using *BSP Tree*



Rendered Examples

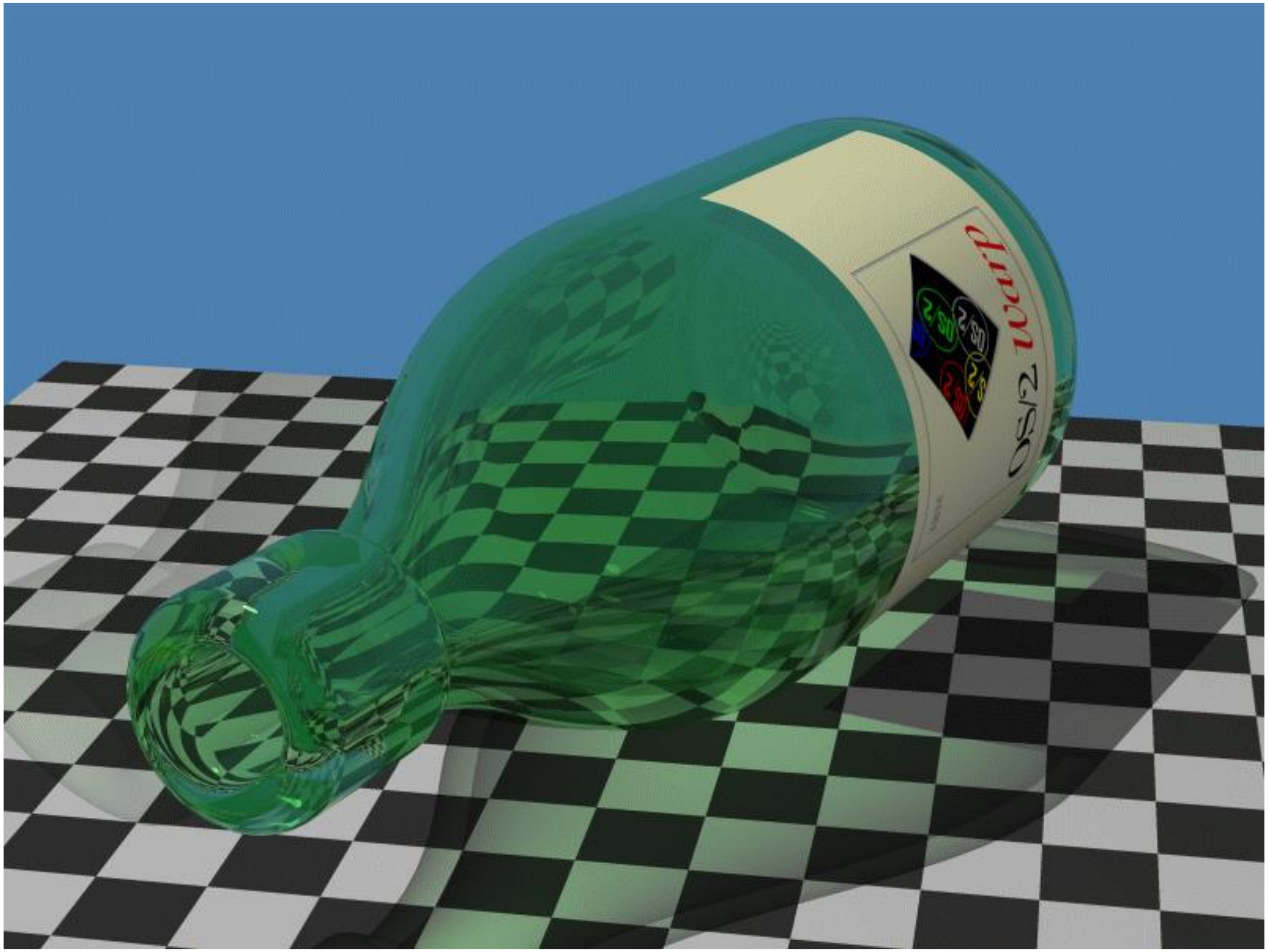


Advanced Phenomena

- Ray tracers can simulate (not always efficiently)
  - Soft shadows
  - Fog
  - Frequency dependent light

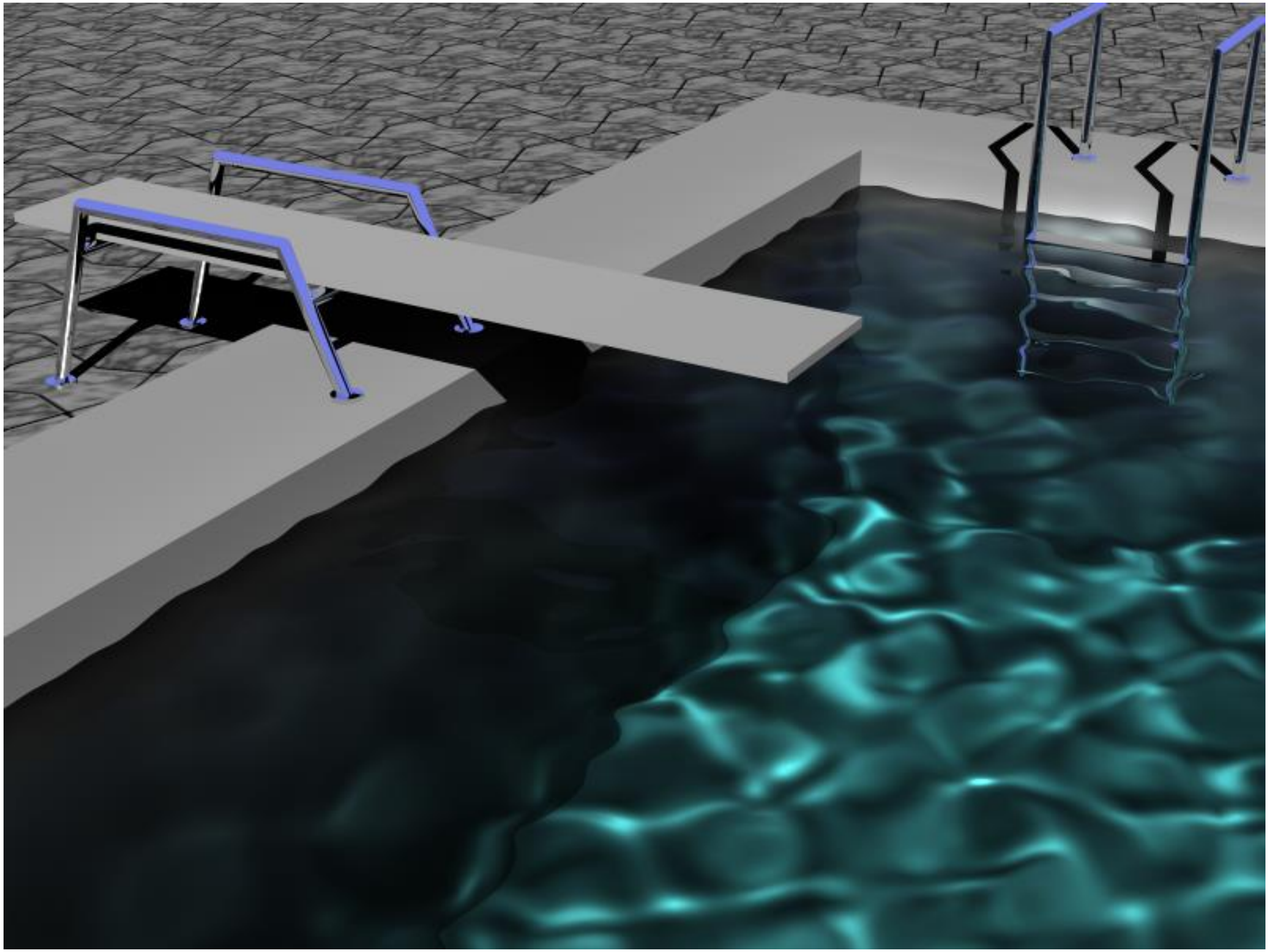
    (Snell's law is different for different wave-lengths)
- But can barely handle diffuse/ambient lighting
  - *Radiosity* is a global illumination scheme complementing ray-tracing for diffuse/ambient lighting

Examples



(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples

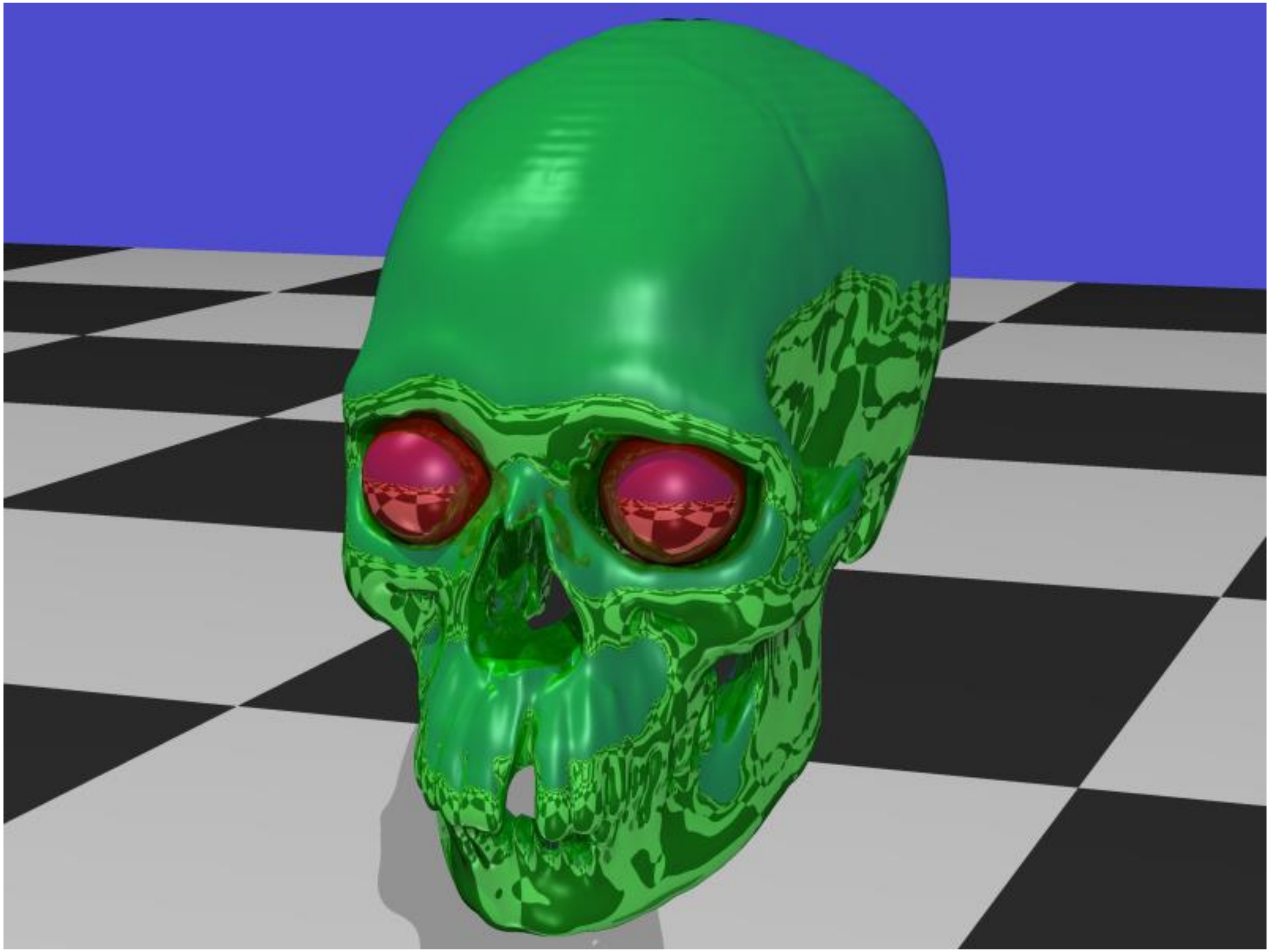


(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples

(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples



(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Summary

- What is ray casting? What are its advantages and disadvantages?
- What is ray tracing? What are its advantages and disadvantages?
- How can we compute the rays through raster points for ray tracing? How can we compute the intersections of such a ray and a plane or a sphere? How is this done for other shapes?
- How can ray tracing be accelerated?