Today’s Topics

11. Texture mapping
12. Introduction to ray tracing

Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
Texture Mapping: Motivation

Goal: Endow objects with more varied & realistic appearance through complex variations in reflectance

Key features of texture mapping
- efficient to render
- reusable
- can modulate albedo and/or fine geometry
Introduction to Texture Mapping

Basic questions:
1. Where do textures come from?
2. How do we map textures onto surfaces?
3. How can textures be used to control appearance?
4. How do we integrate texture mapping and scan conversion?

Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
### Photos of real materials

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Felt</td>
<td>2</td>
<td>Polyester</td>
<td>3</td>
<td>Terry Cloth</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>Pebbles</td>
<td>9</td>
<td>Frosted Glass</td>
<td>10</td>
<td>Plaster a</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>Aluminum Foil</td>
<td>16</td>
<td>Cork</td>
<td>17</td>
<td>Rough Tile</td>
<td>18</td>
</tr>
<tr>
<td>22</td>
<td>Lambrinth</td>
<td>23</td>
<td>Lettuce Leaf</td>
<td>24</td>
<td>Rabbit Fur</td>
<td>25</td>
</tr>
<tr>
<td>29</td>
<td>(2 zoomed)</td>
<td>30-111</td>
<td>zoomed</td>
<td>31-12</td>
<td>zoomed</td>
<td>32-14</td>
</tr>
</tbody>
</table>

*Dana et al., TOG-99*

### Procedurally defined textures

![Teapot Image]
Texture Synthesis

Original Synthesized

(kwatra et al., SIGGRAPH '05)

Texture Synthesis

Original Synthesized

(kwatra et al., SIGGRAPH '05)
Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
Texture Coordinates

Key ideas
- define a 2D coordinate system for texture image (called texture coordinates)
- define a mapping from triangles to texture coards

Specifying a Triangle’s Texture Coordinates

Two ways to define the mapping:
1. for each face of mesh, specify the texture coordinates of each vertex of the face
2. define a continuous mapping from surface pts to texture pts
Texture Coords from Triangle Vertices

Two ways to define the mapping:
1. For each face of mesh, specify the texture coordinates of each vertex of the face
2. Define a continuous mapping from surface pts to texture pts

Texture Coords from Surface Parameters

Two ways to define the mapping:
1. For each face of mesh, specify the texture coordinates of each vertex of the face
2. Define a continuous mapping from surface pts to texture pts
Example: Mapping Textures on a Sphere

Texture coordinates of \( s(\alpha, \beta) \):

\[
\begin{align*}
    u & = \frac{\alpha}{2\pi} \\
    v & = \frac{\beta}{\pi}
\end{align*}
\]

\[
s(\alpha, \beta) = \begin{bmatrix}
x_0 + r\cos \alpha \sin \beta \\
y_0 + r\sin \alpha \sin \beta \\
z_0 + r\cos \beta
\end{bmatrix}
\]

\( \alpha \in [0, 2\pi) \)
\( \beta \in [0, \pi) \)

Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
Appearance Control via Texture Mapping

Approach #1 (modulation)
\[
\bar{F}_d = I \cdot \bar{r}_d, \quad \bar{F}_a = I \cdot \bar{r}_a, \quad \bar{F}_s = I \cdot \bar{r}_s
\] (assumes I normalized to range \([0,1]\))

Approach #2 (replacement)
\[
\bar{F}_d = I, \quad \bar{F}_a = I, \quad \bar{F}_s = I
\]
Appearance Control via Texture Mapping

Texture image

Intensity of texture pixel

(\text{cafe of Phong model})

Other options also possible - see \texttt{g1TexEnvf()} in OpenGL

Topic 11:
Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
Texture Coords of Pixels in Triangle’s Interior

Q: how do we determine the texture coordinates of interior polygon pixels during scan-conversion?

Reminder: Basic Scan Conversion

Step a:
for each edge, interpolate d, L,... between its two vertices

Step b:
for each scanline, interpolate d, L,... between two edge pixels
Reminder: Basic Scan Conversion

**Step a:**
for each edge, interpolate \( d_i, L_i, ... \) between its two vertices.

**Step b:**
for each scanline interpolate \( d_i, L_i, ... \) between two edge pixels.

---

Should we Interpolate Texture Coordinates?

**Q:** Will this interpolation scheme also work for texture coordinates?

**Not very well!**
Artifacts Caused by Naive Scan Conversion

Why Do These Artifacts Occur?

Need to consider the following questions:

- which 3D point \( \overrightarrow{p} \) projects to pixel \( \overrightarrow{q} = (x_i, y_i) \)?
- what are the texture coords of point \( \overrightarrow{p} \)?
Why Do These Artifacts Occur?

\[ \text{Ans: } \frac{1}{2} \left[ \frac{1}{2} (u_1, v_1) + \frac{1}{2} (u_3, v_3) \right] + \frac{1}{2} \left[ \frac{1}{2} (u_1, v_1) + \frac{1}{2} (u_3, v_3) \right] \]

\[ \cdot \text{ Suppose } \overline{q} \text{ is the midpoint between } \frac{1}{2} (q_1 + q_2) \text{ and } \frac{1}{2} (q_2 + q_3) \]

\[ \cdot \text{ What texture coords will be assigned to } \overline{q}? \]
Why Do These Artifacts Occur?

\[
\text{Ans: } \overline{P} = \frac{1}{2} \left( \frac{1}{2} (\overline{P_1} + \overline{P_2}) \right) + \frac{1}{2} \left( \frac{1}{2} (\overline{P_2} + \overline{P_3}) \right)
\]

\[
= \frac{1}{2} \left[ \frac{1}{2} (u_1, v_1) + \frac{1}{2} (u_3, v_3) \right] + \frac{1}{2} \left[ \frac{1}{2} (u_3, v_3) + \frac{1}{2} (u_2, v_2) \right]
\]

- suppose \( \overline{\alpha} \) is the midpoint between \( \frac{1}{2} (\overline{q_1} + \overline{q_2}) \) and \( \frac{1}{2} (\overline{q_2} + \overline{q_3}) \)

- problem: \( \overline{P} \) will not project to \( \overline{\alpha} \) in general

- perspective projection does NOT preserve ratios!

- problem: \( \overline{P} \) will not project to \( \overline{\alpha} \) in general
Reducing Artifacts of Naive Scan Conversion

The straight line $\overline{P_2P_4}$ on the polygon will appear "broken" in the projected triangle.
Reducing Artifacts of Naive Scan Conversion

It is possible to reduce the magnitude of distortions by subdividing the polygon.

Reducing Artifacts by Polygon Subdivision

It is possible to reduce the magnitude of distortions by subdividing the polygon.
Reducing Artifacts by Polygon Subdivision

It is possible to reduce the magnitude of distortions by subdividing the polygon.

8 polys

Reducing Artifacts by Polygon Subdivision

It is possible to reduce the magnitude of distortions by subdividing the polygon.

16 polys
Reducing Artifacts by Polygon Subdivision

It is possible to reduce the magnitude of distortions by subdividing the polygon.

Topic 11:
Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping
Perspectively-Correct Texture Mapping

Main Idea #1:
The mapping from polygon points to their projection is a homography $H_1$, most general transform that maps 2D lines to 2D lines.

Main Idea #2:
The mapping from polygon points to their texture coordinates is a homography $H_2$, most general transform that maps 2D lines to 2D lines.
Perspectively-Correct Texture Mapping

\[ H \tilde{q}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \quad i = 1, \ldots, m \]

1. Compute the homography \( H \) such that

2. Use \( H \) to compute the texture coords of a pixel \( \tilde{q} = (x', y', 1) \) in homogeneous coords
Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

Aliasing During Texture Mapping

Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies").
Aliasing During Texture Mapping

Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies").
MIP-Mapping: Basic Idea

Solution: Use high-resolution texture for rendering objects that are close & low-res (i.e. blurred) texture when they are far away

We can use textures that perturb normals instead of colors or reflectances
Environment Mapping

- Place the center of projection inside a sphere or cube.
- Texture-map the sphere/cube interior with a photo.
- Rendered images now correspond to views of the environment where the photo was taken.
- Particularly effective technique for rendering reflective objects.

Environment Mapping

To create a full 360-degree environment map, we must texture all 6 interior faces of the cube.
Environment Mapping

[Diagram of environment mapping with colors and images]

Environment Mapping

[Diagram of environment mapping with labels and images]

Image from slides by:
Topic 12:
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction
Rasterization vs. Ray Tracing

**Rasterization:**
- Project geometry onto image
- Compute pixel color using local shading model

**Ray tracing:**
- Project pixels (aka *image samples*) backwards onto scene
- Compute pixel color at *q* by estimating light reaching *p* directly or indirectly
Ray Tracing: Basic Idea

Rasterization:

- Project geometry onto image
- Compute pixel color using local shading model

Ray tracing:

- Project pixels (aka "image samples") backwards onto scene
- Compute pixel color at \( \mathbf{q} \) by estimating light reaching \( \mathbf{P} \) directly or indirectly
  - done by recursively casting rays from \( \mathbf{P} \) to possible incident directions

Rasterized

Ray traced
Ray Tracing: Advantages

- Highly customizable (plug-ins for reflectance models, ray sampling functions)
- Can model shadows, arbitrary reflections (e.g., mirrors), refractions, indirect illumination, sub-surface scattering, ...
- Parallelizable
- Allows trading off speed for accuracy (through #cast rays)

Ray Tracing: Basic Algorithm

Basic loop:

- For each pixel \( \bar{q} \)
  - Cast ray \( r \) through \( \bar{q} \)
  - Find 1st intersection of \( \bar{q} \) with scene (i.e., point \( \bar{p} \))
  - Estimate amount of light reaching \( \bar{p} \)

- Estimate amount of light travelling from \( \bar{p} \) to \( \bar{q} \) along ray \( r \)
Ray Tracing: Basic Algorithm

Basic loop:
for each pixel $\bar{q}$
  0. cast ray $r$ through $\bar{q}$
  1. find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)
  2. estimate amount of light reaching $\bar{p}$
     a. *spawn* rays $r_1, r_2, ..., r_k$ from $\bar{p}$ in various directions
     b. if ray $r_i$ hits a light source, estimate light travelling along $r_i$ and stop
     c. else apply loop recursively to ray $r_i$
  3. estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray $r$
Online Ray Tracing Competitions

www.irtc.org/stills

380K triangles, 104 lights, full global illumination in real time

SIGGRAPH '05 Course by Sisalteck et al
Computational Issues in Basic Ray Tracing

Basic loop:

for each pixel \( \vec{q} \)

1. cast ray \( \vec{r} \) through \( \vec{q} \)
2. find 1st intersection of \( \vec{q} \) with scene (i.e. point \( \vec{p} \))
3. estimate amount of light reaching \( \vec{p} \)
   a. "spawn" rays \( r_1, r_2, \ldots, r_n \)
      from \( \vec{p} \) in various directions
   b. if ray \( r_i \) hits a light source, estimate light travelling along \( r_i \) and stop
   c. else apply loop recursively to ray \( r_i \)
4. estimate amount of light travelling from \( \vec{p} \) to \( \vec{q} \) along ray \( \vec{r} \)
   evaluating reflectance model at \( \vec{p} \)

computing ray-scene intersections

defining the ray \( \vec{r} \)
Ray Tracing: Basic Algorithm

Basic loop:

for each pixel $\bar{q}$

0. cast ray $r$ through $\bar{q}$

1. find 1st intersection of
   $\bar{q}$ with scene (i.e. point $\bar{p}$)

2. estimate amount of light
   reaching $\bar{p}$

   a. spawn rays $r_1, r_2, \ldots, r_k$
      from $\bar{p}$ in various
      directions
   b. if ray $r_i$ hits a light
      source, estimate light
      travelling along $r_i$ and stop
   c. else apply loop recursively to ray $r_i$

3. estimate amount of light travelling from
   $\bar{p}$ to $\bar{q}$ along ray $r$
Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Computing the Ray Through a Pixel

Basic loop:

for each pixel $\Bar{q}$
  1. cast ray $\Bar{r}$ through $\Bar{q}$
  2. find 1st intersection of $\Bar{r}$ with scene (i.e. point $\Bar{p}$)
  3. estimate amount of light reaching $\Bar{p}$
  4. estimate amount of light travelling from $\Bar{p}$ to $\Bar{q}$ along ray $\Bar{r}$

Compute ray $\Bar{r}$ given
- discrete pixel position
- world-to-camera matrix $M_{wc}$
- camera-to-canonical view matrix $M_{cv}$
Computing the Ray Through a Pixel: Main Idea

Idea: ray through $\bar{q}$ contains the points $\bar{c} + \alpha(\bar{q}_w - \bar{c}) \quad \alpha \in \mathbb{R}$

where $\bar{q}_w$ are the world coordinates of pixel $\bar{q}$

Compute ray $r$ given
- discrete pixel position
- world-to-camera matrix $M_{wc}$
- camera-to-canonical view matrix $M_{cv}$

Computing the Ray Through a Pixel: Steps

Computing homogeneous coords of $\bar{q}_w$:

1. Convert discrete pixel coordinates (row & column number) to canonical coordinates $(x_i, y_i)$ that lie in the range $[-1, 1]$

Compute ray $r$ given
- discrete pixel position
- world-to-camera matrix $M_{wc}$
- camera-to-canonical view matrix $M_{cv}$
Computing the Ray Through a Pixel: Steps

Computing homogeneous coords of $\vec{q}_w$:

1. Compute canonical view coordinates $(x_c, y_c)$

2. Compute homogeneous 3D canonical view coordinates of $\vec{q}$:
   $$\vec{q}_v = \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$$

Compute ray $r$ given:
- discrete pixel position
- world-to-camera matrix $M_{wc}$
- camera-to-canonical view matrix $M_{cv}$