Today

- Maps
  - Texture Mapping
  - Bump Mapping
  - Displacement Mapping
  - Shadow Maps
  - Environment Maps
- Compositing
Surface Detail

- The real world is complicated
- We can’t explicitly model all the rich detail
- So, we come up with some “hacks”...

Texture Mapping

- The idea is to wrap a “texture” onto a surface
- To do this we need
  - A texture, usually just an image
  - A parameterization of the surface
  - A mapping from the surface parameterization to the texture coordinates
Barycentric Coordinates

- $X$ can be expressed as
- $\alpha P_1 + \beta P_2 + \gamma P_3$
  where
  $\alpha + \beta + \gamma = 1$
- or, alternatively as
  $(1 - \alpha - \beta)P_1 + \alpha P_2 + \beta P_3$

Barycentric Coordinates

- We can use barycentric coordinates to interpolate any quantity (color, texture coordinates, etc) stored at vertices, not just positions.
Bad Idea

- Simplest (and fastest) approach is to compute texture coordinates for polygon vertices and interpolate in screen space.
- This gives the image on the right.

Undoing Homogenization

- Let \( P_i = (x_i, y_i, z_i, h_i) \) be the \( i^{th} \) point of some polygon, after projection, but before homogenization.
- The homogenized point \( S_i = P_i / h \) is the location of \( P_i \) on the screen.
- Let \( X \) be a point we wish to shade, we have its barycentric coordinates in screen space:
  \[
  X = \sum_i b_i S_i
  \]
Undoing Homogenization

- We know \( S_i = P_i/h \)
- We also know that there exist weights \( a_i \), such that
  \[
  X = \left( \sum_i a_i P_i \right) / \left( \sum_j a_j h_j \right)
  \]
- Combining the above we have
  \[
  \sum_i b_i S_i = X = \left( \sum_i a_i P_i \right) / \left( \sum_j a_j h_j \right)
  \]
  \[
  \sum_i b_i (P_i/h_i) = \left( \sum_i a_i P_i \right) / \left( \sum_j a_j h_j \right)
  \]

---

Undoing Homogenization

\[
\frac{b_i}{h_i} = \frac{a_i}{\left( \sum_j a_j h_j \right)} \quad \forall i
\]
\[
\frac{b_i \left( \sum_j a_j h_j \right)}{h_i} = a_i \quad \forall i
\]
\[
\frac{b_i \left( \sum_j a_j h_j \right)}{h_i} - a_i = 0 \quad \forall i
\]

- This is a linear system in \( a_i \)
- Unfortunately is is non-invertible, so...
Undoing Homogenization

- we add
  \[ \sum_i a_i = 1 \quad \sum_i b_i = 1 \]
- now its solvable and the solution is:
  \[ a_1 = \frac{h_2 h_3 b_1}{h_2 h_3 b_1 + h_1 h_3 b_2 + h_1 h_2 b_3} \]
- similar formulas exist for \( a_2 \) and \( a_3 \)

Bump/Displacement Mapping

- Texture mapping changes a surface’s reflectance, but that can’t give us a realistic orange
- For this we can use bump or displacement mapping
Bump Mapping

- The idea is to perturb the surface normals
- If the bump map is an array of vectors, just add the bump vectors to the surface normals
- If the bump map is an array of scalars (desired displacements along the normal direction), then the new normal is
  \[ n' = n + b_u(n \times P_v) - b_v(n \times P_u) \]

Displacement Mapping

- Actually perturb the location of the surface, usually along the normal direction, by scalar values given in the displacement map
- This is usually done by moving the vertices of a polygonal mesh
Bumps vs. Displacements

- Bumps do not cast shadows or change the silhouette, they do produce specular effects
- Displacements actually change the geometry
- Displacement maps only look good on high resolution models
- Bottom line: bumps are cheaper, displacements look better

Shadow Maps

- Key insight: If we render the scene from the point of view of the light source, the lit surfaces will be visible and the unlit surfaces will be hidden
- We render the scene from the point of view of the light source
- Store the z values in a “depth shadow map”
Shadow Maps

- For each polygon
  - Render the polygon from the camera
  - Render the polygon from the light
  - Compare the z value from the light with the one in the depth shadow map
    - If they match, the polygon is lit
    - Otherwise it is in shadow

Environment Maps

- Fake reflections
- Assumes the environment is very far away
- Depends on the location of the camera
- Usually stored in a spherical table or a cube map
Environment Maps

- Remove the reflective object from the scene
- Render the scene six times with the eye at the center of the removed object
- Render the scene, using reflection vectors to index the cube map

Compositing

- Sometimes scenes are too complex to render all at once
- Different parts of a scene often do not interact
- Need a way to render pieces separately and put them back together later
Alpha Channels

- Alpha channel stores opacity
- Primary operation is “over”
- Pre-multiplied alpha allows the use of the same rules for all 4 channels

Normal Alpha Channel
\[ c = \alpha c_f + (1 - \alpha) c_b \]

Pre-multiplied Alpha Channel
\[ c = c_f + (1 - \alpha) c_b \]

Alpha Channel

- Other Operations

\[ c = F c_f + G c_g \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Over</td>
<td>1</td>
<td>1 - ( \alpha_f )</td>
</tr>
<tr>
<td>Inside</td>
<td>( \alpha_g )</td>
<td>0</td>
</tr>
<tr>
<td>Outside</td>
<td>1 - ( \alpha_g )</td>
<td>0</td>
</tr>
<tr>
<td>Atop</td>
<td>( \alpha_g )</td>
<td>1 - ( \alpha_f )</td>
</tr>
<tr>
<td>Xor</td>
<td>1 - ( \alpha_g )</td>
<td>1 - ( \alpha_f )</td>
</tr>
<tr>
<td>Clear</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Set</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Suggested Reading

- Fundamentals of Computer Graphics by Pete Shirley
  - Chapters 10, 3.4