Real Time Reflections

Han-Wei Shen
Reflections

- One of the most noticeable effects of inter-object lighting
- Direct calculation of the physics (ray tracing) is too expensive
- Our focus is to capture the most significant reflection while minimizing the overhead via rendering the “virtual object”
Image vs. Object Space Methods

- **Image space methods**: create a texture from a view of the reflected objects and apply it to the reflector
  - Advantage: does not depend on the object geometry
  - Disadvantage: sampling issue and also only an approximation (environment mapping as an example)

- **Object space methods**: create the actual geometry of the object’s reflection and render it to the reflector
  - Disadvantage: accuracy of the geometry
  - Advantage: more accurate reflection (for nearby objects)

- Both methods need to create the virtual objects
Planar Reflections

- The most common reflection – flat mirror, floor, wall, etc
- Creating virtual objects for the reflected objects is much easier
- A view independent operation – only consider the relative position of the object and the reflector
- The virtual object is created by transforming the object across the reflector plan
Reflection Transformation

- Can be broken into three stages:
  1. Transform the objects into the reflector’s local coordinate system
     - Translate the reflector to the world origin and rotate so the reflector plane will coincide with the world’s XY plane
  2. Scale by -1 in Z
  3. Transform the reflected object’s local coordinates of the reflector back to the world coordinates
     - Rotate the world’s XY plane to the reflector’s plane and then translate the origin to the reflector’s local origin

- Overall - A translation of the mirror plane to the origin, a rotation embedding the mirror to x-y plane, a scale of -1 in Z, inverse rotation, and a translation back to the mirror location
Reflection Matrix

- Given a point P on the reflector plane, vector V perpendicular to the plane

\[
R = \begin{bmatrix}
1 - 2V_x^2 & -2V_xV_y & -2V_xV_z & 2(P \cdot V)V_x \\
-2V_xV_y & 1 - 2V_y^2 & -2V_yV_z & 2(P \cdot V)V_y \\
-2V_xV_z & -2V_yV_z & 1 - 2V_z^2 & 2(P \cdot V)V_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- Multiply R to the object vertices produce an reflected object on the opposite side of the reflector plane

- The entire scene is duplicated, simulating a reflector of infinite extent
Render the Reflected Geometry

- An important task: clip the reflected geometry so that it is only visible on the reflector surface
- Beyond the reflector boundaries and in front of reflector
Clipping using the stencil

- You only want the reflected geometry to appear on the reflector surface
- Use stencil buffer:
  - Clear the stencil buffer
  - Render the reflector and set the stencil
  - Render the reflected geometry only at where the stencil pixels have been set
- The above algorithm is to use the stencil buffer to control where to **draw** the reflection
Clipping using the stencil

- Another method: render the reflected object first, and then render the reflector to set the stencil buffer, then clear the color buffer everywhere except where the stencil being set.

- This method is to use the stencil buffer to control where to erased the incorrect reflection.

- Advantage: when it is faster to use stencil to control clearing the scene than drawing the entire scene with stenfil tests.
The stencil erase algorithm

- Draw the reflected objects
- Clear the stencil and depth buffers
- Configure the stencil so that 1 will be set where polygons are rendered
- Disable draw to color buffer
- Draw the reflector
- Reconfigure the stencil
- Clear the color and depth buffer to the background color (except where the stencil being set)
- Disable stencil test
- Draw the rest of scene (except reflected objects and reflector)
The stencil erase algorithm

Figure 37. Stencil Reflection Steps
Clipping using texture mapping

- Render the reflected geometry
- Store the image into a texture (using FBO or glCopyTexImage2D)
- Clear the color and depth buffer
- Redraw the entire scene, with the reflector textured with the previous texture (the process of texturing reflector will automatically clip the reflected geometry outside the boundary)
- Note that reflected geometry in front of the reflector still needs to be clipped before the texture map is created (using the reflector as the clipping plane - supported by OpenGL’s application clipping plane)
Clipping using texture mapping

- The key is to assign correct texture coordinates to the reflector
- Basic idea – s and t coordinates correlate to x and y window coordinates of the reflector
- Project the reflector vertex to the clip space
- Bias and scale the texture coordinates from [-1,1] to [0,1]
- Perform perspective division and use the first two components to access the texture
Clipping using texture mapping

Figure 58. Masking Reflections Using Projective Texture
Interreflection

- Goal: let the reflections to “bounce” between reflectors
- The number of bounces needs to be limited to preserve interactivity
- The reflection transformations need to be concatenated
- Render the deepest interreflection first
Interreflection

- Clear the stencil buffer
- Set the stencil operation to increment the stencil values where pixels are rendered
- Render each reflector involved in interreflection into stencil buffer
- Set the stencil test to pass where the stencil value equal the number of interreflection
- Apply planar transformation
- Draw the reflected scene
- Draw the reflector, blending if desired
Refraction

- Snell’s law – refraction angle is based on the medium’s refraction index

\[ n_1 \sin \phi_1 = n_2 \sin \phi_2 \]
Refraction Vector

- \( n = \frac{n_1}{n_2} \)

\[
R = nU - N \left( n(N.U) + \sqrt{1-n^2(1-(N.U)^2)} \right) \quad \text{or} \quad R = U - (1-n)N(N.U) \quad \text{(a simplified form)}
\]
Planar Refraction

- Move the eye point to match the refraction vector

\[ \frac{1}{n} \times d \]

away from the interface
(when eye ray is perpendicular)

rotate the eye (when not Perpendicular)

When the refraction ray bent toward the normal direction (a common case)
Environment Map Refraction

- Take 6 shots from the refractor to its environment
- Based on the eye rays, calculate the refracted rays using Snell’s law at each refractor’s vertex
- Using the refracted rays as texture coordinate (s, t, r) to look up the environment cube map
Environment Map Refraction

- Create Environment Cube Map
- Calculate the refractor ray and use it as the texture coordinates to the environment cube map
- Result