Overview

- What are Cube Maps?
- How to Create a Cube Map
- Using Cube Maps for Environment-Maps
- Pre-Calculated Specular & Diffuse Lighting
What Are Cube Environment Maps?

- Cube Maps are made up of 6 square textures of the same size, representing a cube centered at the origin.
- Each cube face represents a set of directions along each major axis.
- \(+X, -X, +Y, -Y, +Z, -Z\)
- Think of a unit cube centered about the origin.
- Each texel on the cube represents what can be ‘seen’ from the origin in that direction.
Visualizing the Cube Map

The Cube map is accessed via vectors expressed as 3D texture coordinates (S, T, R).

The greatest magnitude component, S, T or R, is used to select the cube face. The other 2 components are used to select a texel from that face.
Cube Map Texture Coordinates

The calculation that is performed to generate the coordinates is simply a 3D → 2D projected texture.

1. Select the highest magnitude component, let’s say -T
2. Divide the other components by -T, giving
   
   \[ S' = S / -T \]
   \[ R' = R / -T \]
Creating Cube Maps in OpenGL

- ARB_texture_cube_map & EXT_texture_cube_map

- Defines `<cap>` parameter
  GL_TEXTURE_CUBE_MAP_EXT

- Defines new texture `<target>` parameters for texture functions such as:
  - `glTexImage2D()`
  - `glCopyTexImage2D()`
  - `glCopySubTexImage2D()`
  - `glCopySubTexImage2D()`
  - `...`
Creating Cube Maps in OpenGL

- ARB_texture_cube_map & EXT_texture_cube_map
- Defines new texture coordinate generation modes
  - GL_REFLECTION_MAP_EXT
  - GL_NORMAL_MAP_EXT
Creating Cube Maps in OpenGL

```c
glTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_px );
gTexImage2D( GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_nx );
gTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_py );
gTexImage2D( GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_ny );
gTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_pz );
gTexImage2D( GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT, 0,
             GL_RGB8, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, face_nz );

glEnable( GL_TEXTURE_CUBE_MAP_EXT );
/* Render geometry. */

glDisable( GL_TEXTURE_CUBE_MAP_EXT );
```
Environment Mapping with Cube Maps

- Reflections of environment on shiny objects

- Other techniques
  - Exhibit interpolation artifacts
  - More difficult to generate for dynamic scenes

- Cube Environment maps
  - Easy to generate on the fly
  - $S,T,R$ can be automatically calculated in HW
  - Improved interpolation
Generating an Environment Map

- Set up a 90° FOV camera at the object’s location
- Point the camera along +X, and render the (approximate) scene around your object into the first face of the cube map
- Repeat the process, facing -X, ± Y and ± Z into each face of the cube map
- You now have a dynamically generated environment map!
Example of Cube Environment Mapping
Optimizations

- Dynamic Cube Maps don’t have to be full resolution
- They also don’t need updating every frame
- Allowing updates to lag behind a frame prevents stalling in some cases
Cube Map for Pre-Computed Specular lighting (Illumination map)

- Use a cube map as a specular lighting solution for reflective objects:
  - The cubemap can be thought of as a function of a vector that returns a RGBA value.
  - Therefore, any lighting that relies on only a vector and constant values can be precalculated and stored in the cube map.
  - Render only your specular lighting into your cube map (only valid for the current view vector).
  - Use camera space reflection texture coordinate generation.
  - Use blurred or low resolution cube map for rough surfaces.
Cube Map for Diffuse Lighting

- To use the environment map as a diffuse lighting solution for diffuse objects:
  - Render only diffuse lighting into cubemap
  - Use GL_TEXTURE_GEN_MODE
    GL_NORMAL_MAP_EXT
  - Works well for directional (and distant) lights
  - Does not work well when light is very close to surface.
Dynamic Cube Maps

- Cube map only needs to be updated when surrounding lights change
  - Can use the cube map to store pre-calculated lighting (i.e. lightmaps), and add in other lights on top

- Don’t typically have to update the cube map every frame
  - roughly represent the environment, it will look good
Cube Maps as Vector Lookups

- Think of a cube map as a way to store/lookup a function with a vector
  - The function can store a color or a vector
  - Can also store an alpha

- For example, a cube map can store color which represents a normalized vector
  - Allows vector interpolation with normalization
  - \( V' = \text{Normalize}(V) \) on a per-pixel basis
  - Useful for bump mapping and per-pixel lighting
  - Called a *Normalization Cube Map*
The Normalization Cube Map

The Cube map is accessed via vectors expressed as 3D texture coordinates (S, T, R).

The Orange Vector <0, 0, -8> is passed in. The Blue Vector <0, 0, -1> is returned in RGB form as <0x80, 0x80, 0x00>. 
Cube Maps as Vector Lookups

- Other useful functions
  - $V' = -V$
  - Color = $(L \cdot N)$
  - Color = $(R \cdot V)^n$
Questions, comments, feedback

- Chris Wynn, cwynn@nvidia.com
- Sim Dietrich, sdietch@nvidia.com
- www.nvidia.com/developer