



*n*VIDIA™

## **Cube Maps**

**Sim Dietrich**

**NVIDIA Corporation**

**[sim.dietrich@nvidia.com](mailto:sim.dietrich@nvidia.com)**

# Overview

---

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

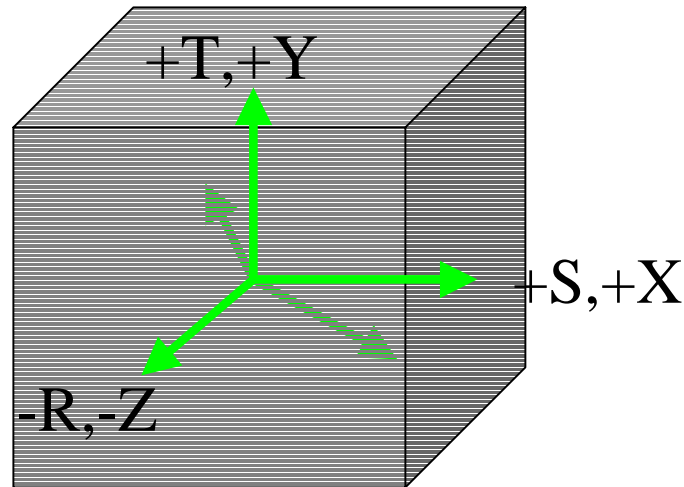
# What Are Cube Environment Maps?

---

- **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  $S' = S / -T$ ,  $R' = R / -T$

# Cube Maps as Reflection Maps

---

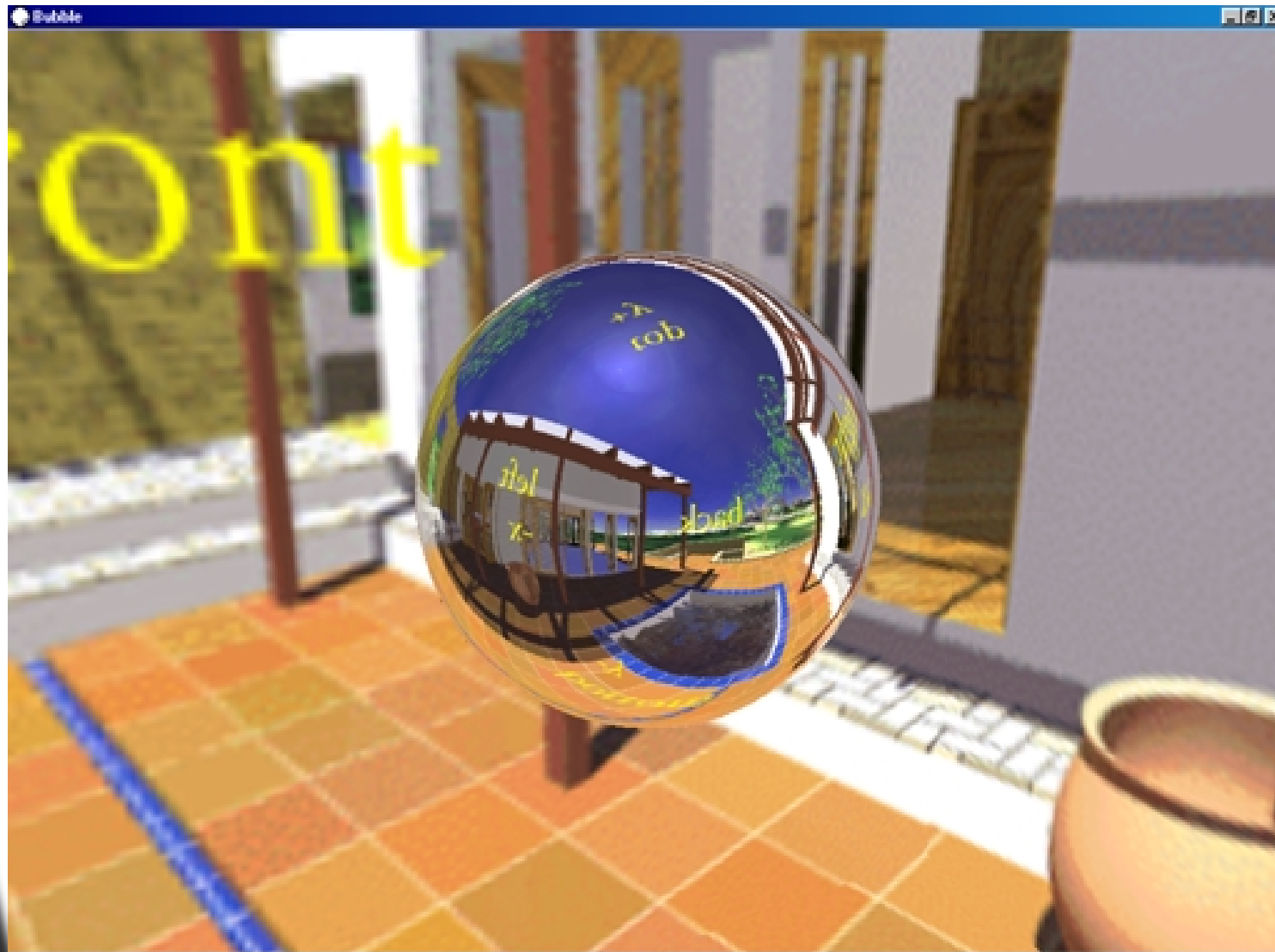
- Shiny objects that reflect their environment add immersiveness and flash to games
- The bad news is that current techniques ( spherical and parabolic environment maps ) can be expensive to generate and use
  - Per-vertex CPU calculation costs for UV coords
  - Spherical maps are view-dependent, must be regenerated to avoid artifacts
- Cube Environment maps solve both of these problems
  - S,T,R can be automatically calculated in HW
  - Easy to generate on the fly to reflect the actual environment

# Generating an Environment Map

---

- **Set up a 90 degree FOV camera at the object's location**
- **Point the camera down +X, and render the (approximate) scene around your object into the first face of the cube map**
- **Repeat the process, facing -X,  $\pm Y$  and  $\pm Z$  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**
- **Also, they could be 16 bit, even if main color buffer is 32 bit**
- **They also don't need updating every frame**
- **Allowing updates to lag behind a frame prevents stalling in some cases**

# Optimizations

---

- You can save off 6 color & z buffers for the static parts of your scene
  - One for each cube face
- Have 1 dynamic Z buffer to use when re-rendering one or more faces
- Each frame,
  - Attach the dynamic Z buffer to the cube face
  - Blt the saved static Z buffer over the dynamic Z buffer
  - Render only dynamic objects into the cube face

# Environment Map as Specular lighting (Illumination map)

---

- To use the environment 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

# Environment Map for Diffuse Lighting

---

- To use the environment map as a diffuse lighting solution for diffuse objects :
  - Render only your diffuse lighting into your cubemap
  - Use normal vector texture coordinate generation
    - D3DTCI\_CAMERASPACE\_NORMAL
  - Works especially well for directional lights, because cube maps are position independent

# Dynamic Cube Maps

---

- **Cube map only needs to be updated when surrounding lights change**
  - **Can use the cube map to store pre-calculated lighting ( ala lightmaps ), and add in other lights on top**
- **Don't typically have to update the cube map every frame**
- **A little cheating works: if you roughly represent the environment, it will look good**

# Creating the Cube Environment Map

---

The cube environment map is represented by a DirectDraw surface with the

**DDSCAPS2\_CUBEMAP** and  
**DDSCAPS\_TEXTURE** flags

Can be marked as **DDSCAPS\_3DDEVICE** for  
dynamic cube maps

Can have mip-maps ( recommended for big  
CEMs )

The cube map can have from 1 to 6 faces

Can leave out faces to save texture memory (   
like the bottom (-Y) )

Can attach a Z buffer to the cube map for  
dynamic generation

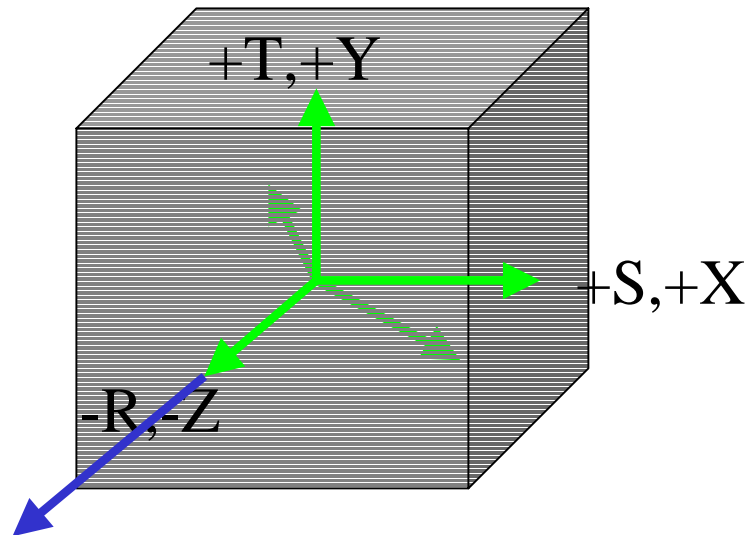
## Cube Maps as Vector Lookups

---

- You can think of a cube map as a way to store a function look-up table indexed by a vector
  - The function can return either a color or a vector, plus an alpha channel
- For instance, a Cube Map can perform  $V' = \text{Normalize}(V)$ 
  - A Normalization Cube Map is useful for Per-pixel lighting and DOT3 Bump mapping
- Or  $V' = -V$
- $\text{Color} = (L \text{ DOT } N)$

# Visualizing the Vector Normalization Cube Map

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



The Blue Vector  $\langle 0, 0, -8 \rangle$  is passed in.

The Green Vector  $\langle 0, 0, -1 \rangle$  is returned in RGB form as  $\langle 0x80, 0x80, 0x00 \rangle$  or  $0xff808000$





**Questions?**

---