418341 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 14

ประมุข ขันเงิน

pramook@gmail.com

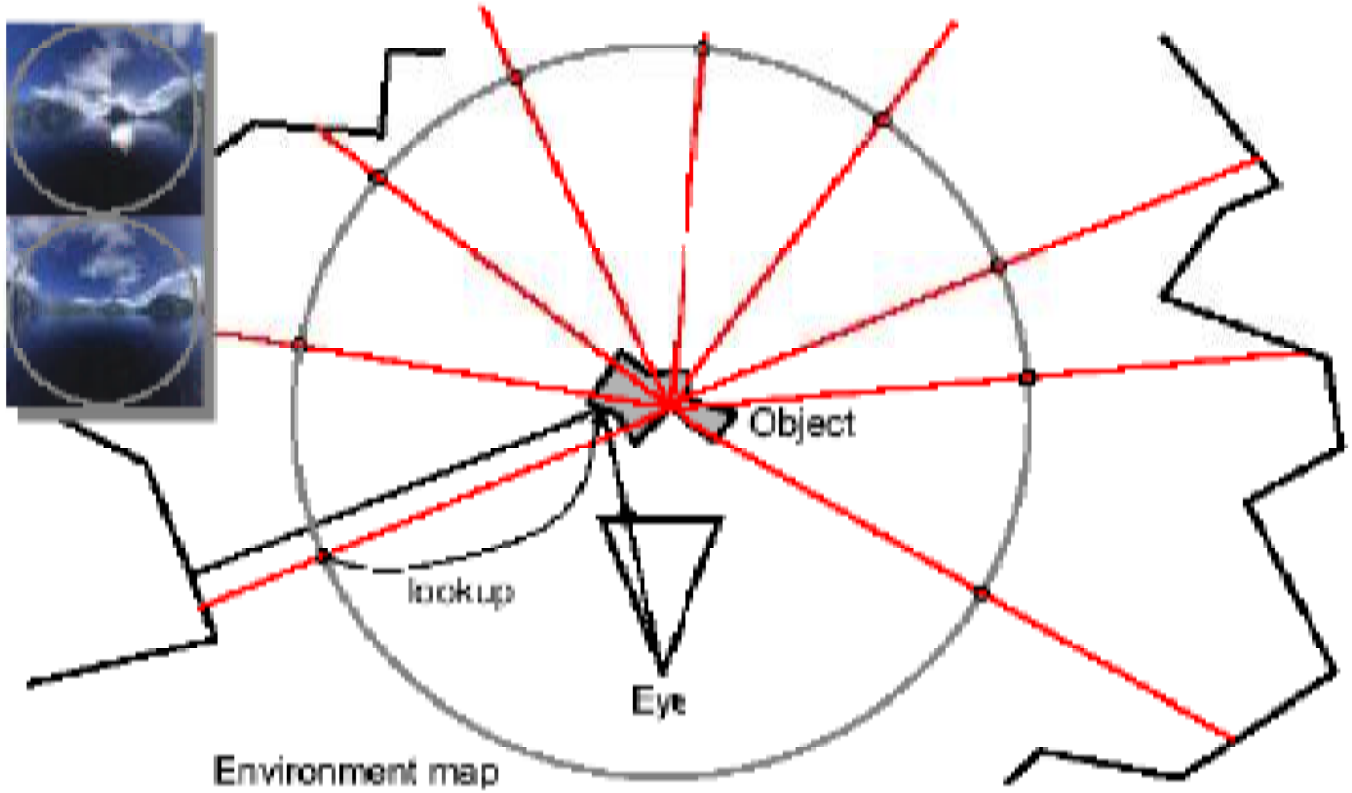
Environment Map

- การใช้ texture เก็บแสงที่พุ่งจาก “สิ่งแวดล้อม” เข้าหาวัตถุในทิศทางต่างๆ

- สมมติว่าวัตถุเป็น “จุด”

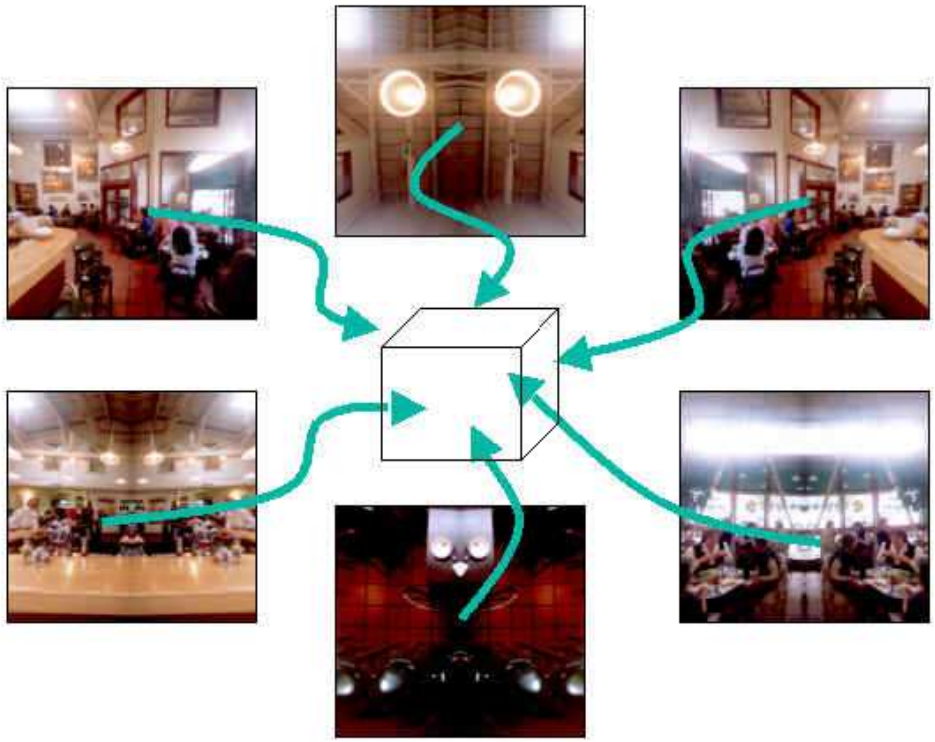
- Texture ใช้ในการตอบคำถามว่า “แสงที่พุ่งเข้าหาวัตถุในทิศทาง (x,y,z) มีสีอะไร”

Environment Map



Cube Map

- เป็นวิธีเก็บ **environment map** แบบหนึ่ง
- ใช้ภาพหกภาพมาประกอบกันเป็นลูกบาศก์



Cube Map ใน OpenGL

- เราสามารถใช้ cube map ใน OpenGL ได้ตั้งแต่ OpenGL เวอร์ชัน 1.3
- ต้องใช้ extension ชื่อ EXT_texture_cube_map
- หมายความว่าเวลาเขียนโปรแกรมใน Windows จะต้องใช้ GLEW

การสร้าง Cube Map

- มีขั้นตอนคล้ายกับการสร้าง texture ธรรมดา
  - Enable การใช้ cube map
  - สร้าง handle ของ cube map ด้วย glGenTextures
  - ทำการ bind cube map ที่สร้างขึ้น
  - Download รูปที่ใช้ทำ cube map ลงสู่ GPU

Enable การใช้ Cube Map

- ให้สั่ง

```
glEnable(GL_TEXTURE_CUBE_MAP_EXT);
```

- และอย่าลืมสั่ง

```
glDisable(GL_TEXTURE_CUBE_MAP_EXT);
```

ก่อนใช้งาน texture แบบอื่น

การสร้าง Handle ของ Cube Map

- เช่นเดียวกับการสร้าง texture อื่นเราต้องประกาศตัวแปรประเภท GLunit เพื่อใช้เก็บชื่อของ cube map

```
GLuint cubeMap;
```

- หลังจากนั้นใช้ glGenTextures สร้าง texture ตามปกติ

```
glGenTextures(1, &cubeMap);
```

Bind Cube Map ที่สร้างขึ้น

- สั่ง glBindTexture โดยใช้ target เป็น GL_TEXTURE_CUBE_MAP_EXT

```
glBindTexture(GL_TEXTURE_CUBE_MAP_EXT,
    cubeMap)
```

Download รูป

- ใช้คำสั่ง glTexImage2D หรือ gluBuild2DMipmaps เพื่อ download รูปเช่นเดิม แต่เราต้อง download รูปจำนวนทั้งหมด 6 รูปสำหรับ 6 ด้านของลูกบาศก์

- เราสามารถระบุว่าจะ download รูปของด้านไหนได้ด้วยการระบุ target ของคำสั่งทั้งสองดังในสไลด์หน้าต่อไป

Target สำหรับ Download รูป

Target	ด้าน
GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT	ขวา
GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT	ซ้าย
GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT	บน
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT	ล่าง
GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT	หน้า
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT	หลัง

ตัวอย่างโค้ด

- ผมสร้างฟังก์ชัน loadCubeMapSide ไว้สำหรับ download รูปเข้าไปยังด้านหนึ่งของ cube map โดยกำหนด
  - Target ที่จะ download รูปลงไป
  - ชื่อไฟล์ของรูปนั้น

- loadCubeMapSide ใช้ DevIL ในการดึงข้อมูลรูปออกมาจากไฟล์

loadCubeMapSide

```
void loadCubeMapSide(GLuint target,
   const char *imageName)
{
   ILuint image;
   ilGenImages(1, &image);
   ilBindImage(image);
   ilLoadImage((wchar_t *)imageName);
   ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE)
   gluBuild2DMipmaps(target,
       ilGetInteger(IL_IMAGE_BPP),
       ilGetInteger(IL_IMAGE_WIDTH),
       ilGetInteger(IL_IMAGE_HEIGHT),
       ilGetInteger(IL_IMAGE_FORMAT),
       GL_UNSIGNED_BYTE,
       ilGetData());
   ilDeleteImages(1, &image);
}
```

โค้ดตัวอย่าง

- ผมเขียนฟังก์ชัน initCubeMap เพื่อ load รูปทั้งสำหรับทั้ง 6 ด้าน

- ใน initCubeMap ผมเรียก loadCubeMapSide เป็นจำนวนหกครั้งเพื่อ download รูป

initCubeMap

```
void initCubeMap()
{
    glEnable(GL_TEXTURE_CUBE_MAP_EXT);
    glGenTextures(1, &cubeMap);
    glBindTexture(GL_TEXTURE_CUBE_MAP_EXT, cubeMap);

    glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MIN_FILTER,   GL_LINEAR_MIPMAP_LINEAR);

    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT, "../images/cm_right.jpg");
    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT, "../images/cm_left.jpg");
    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT, "../images/cm_top.jpg");
    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT, "../images/cm_bottom.jpg");
    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT, "../images/cm_front.jpg");
    loadCubeMapSide(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT, "../images/cm_back.jpg");

    glDisable(GL_TEXTURE_CUBE_MAP_EXT);
}
```

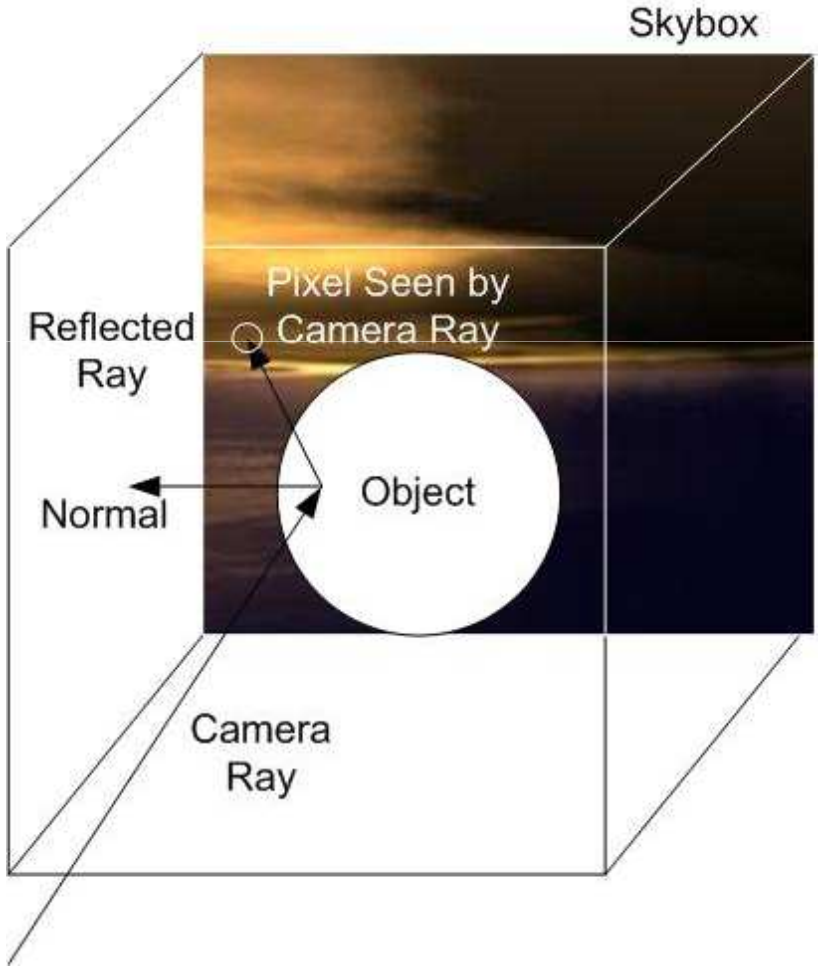
การนำ Cube Map ไปใช้งาน

- กระจก = การสะท้อนแสง
- แก้วใส = การหักเหแสง

หลักการสร้างกระจก

- สร้าง “skybox” หรือกล่องที่มีรูปท้องฟ้า ล้อมรอบวัตถุไว้
- สำหรับ fragment แต่ละ fragment ให้คำนวณทิศทางที่แสงที่เดินทางจากตาไปยังตำแหน่งของ fragment แล้วสะท้อนออกไป
- ให้นำทิศทางที่ได้ไปอ่านข้อมูลจาก cube map

หลักการสร้างกระจก



การสร้างกระจกใน OpenGL

- โดยปกติแล้วเวลาจะอ่านข้อมูลจาก cube map เราจะต้องใช้ texture coordinate 3 ตัว เนื่องจากทิศทางเป็นทิศทางในสามมิติ

- เราสามารถกำหนดทิศทางได้เองด้วยคำสั่ง glTexCoord3d

- แต่ละ component ของทิศทางจะมีค่าตั้งแต่ -1 ถึง 1
  - ไม่ใช้ 0 ถึง 1 เหมือนกับ texture coordinate อื่นๆ

การสร้างกระจกใน OpenGL

- Texture coordinate ที่จะใช้มีสามตัวคือ s, t, และ r

- เราสามารถสั่งให้ OpenGL สร้าง texture coordinate ให้โดยอัตโนมัติได้ด้วยคำสั่ง

  glEnable(GL_TEXTURE_GEN_?)

  - ถ้าอยากให้สร้าง s ให้โดยอัตโนมัติก็สั่ง

    glEnable(GL_TEXTURE_GEN_S);

  - ถ้าอยากให้สร้าง t ให้โดยอัตโนมัติก็สั่ง

    glEnable(GL_TEXTURE_GEN_T);

การสร้างกระจกใน OpenGL

- นอกจากนี้ยังต้องบอกด้วยว่าจะให้สร้าง texture coordinate ให้แบบใด ด้วยคำสั่ง glTexGeni

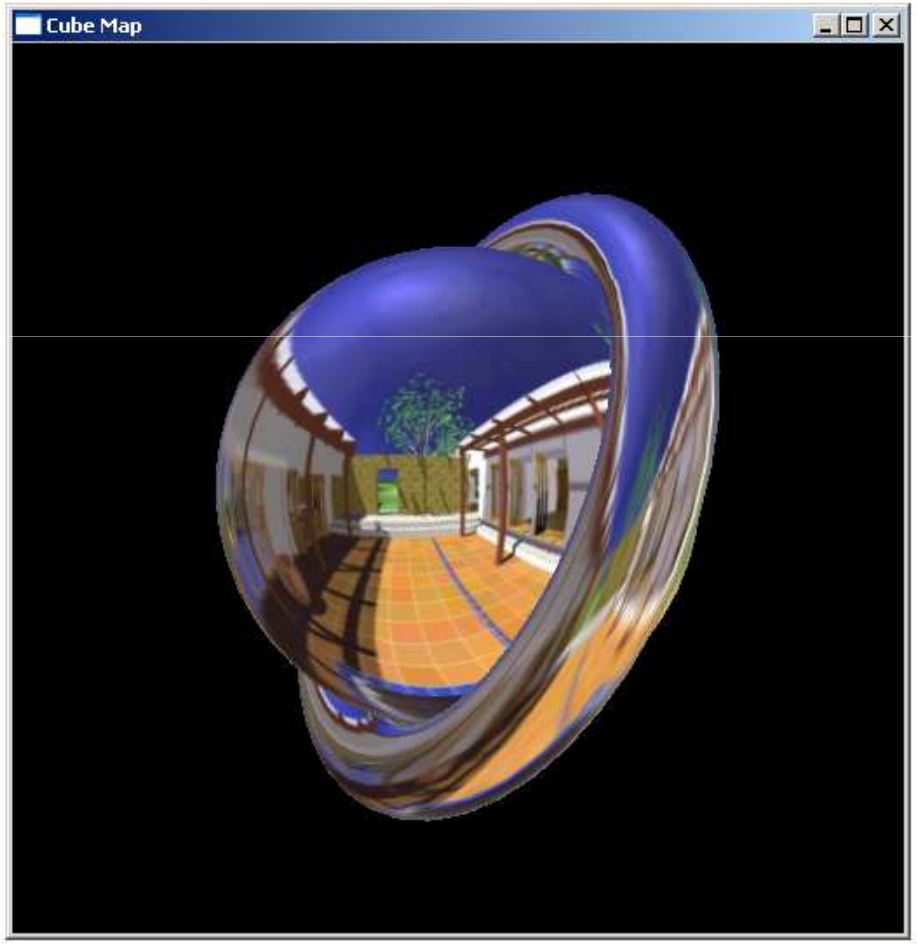
- ในกรณีการสร้างกระจกเราต้องสั่ง

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
  GL_REFLECTION_MAP_EXT);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
  GL_REFLECTION_MAP_EXT);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE,
  GL_REFLECTION_MAP_EXT);
```

ตัวอย่างโค้ด

```
void display()
{
   glEnable(GL_TEXTURE_CUBE_MAP_EXT);

   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
        GL_REFLECTION_MAP_EXT);
   glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
        GL_REFLECTION_MAP_EXT);
   glTexGeni(GL_R, GL_TEXTURE_GEN_MODE,
        GL_REFLECTION_MAP_EXT);

   glEnable(GL_TEXTURE_GEN_S);
   glEnable(GL_TEXTURE_GEN_T);
   glEnable(GL_TEXTURE_GEN_R);

   glutSolidSphere(1.5, 50, 50);

   glutSwapBuffers();
}
```

ดู demo



การใช้ Cube Map ใน GLSL

- GLSL ให้ผู้ใช้สามารถประกาศ uniform parameter ประเภท samplerCube ใน shader ได้ เช่น

```
uniform samplerCube env_map;

void main()
{
      :
      :
}
```

การใช้ Cube Map ใน GLSL

- เวลาอ่านข้อมูลจาก cube map ให้ใช้คำสั่ง texureCube โดย
  - Parameter ตัวแรกเป็นตัวแปรประเภท samplerCube
  - Parameter ตัวที่สองเป็นค่าประเภท vec3

- ตัวอย่าง

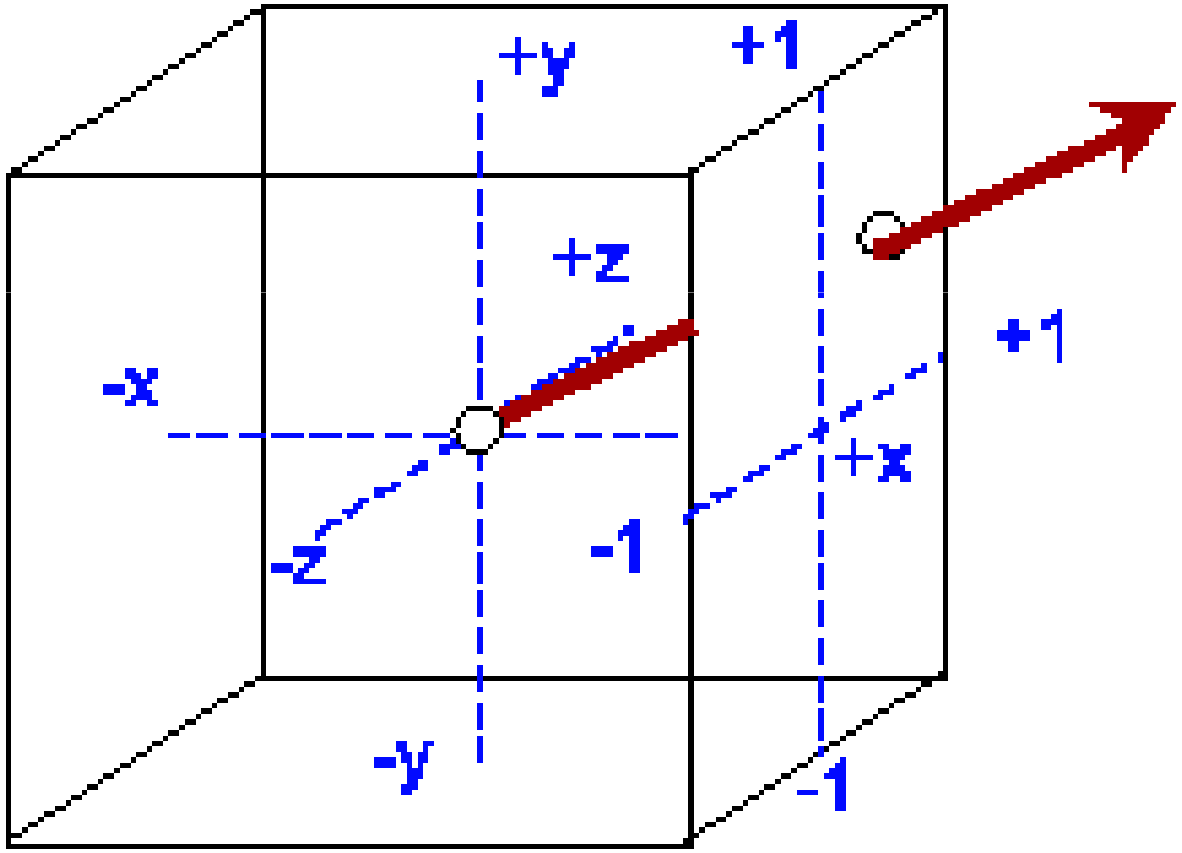
```
color = textureCube(env, vec3(1,0,0));
```

การใช้ Cube Map ใน GLSL

- ตัวแปรประเภท vec3 ที่เราให้ไปต้องเป็นเวกเตอร์หนึ่งหน่วยที่แต่ละมิติมีค่าอยู่ในช่วง [-1,1]

- ค่าที่อ่านได้จาก cube map คือค่าของสีที่จุดที่เกิดจากการยิงรังสีจากจุด (0,0,0) ไปในทิศทางที่กำหนดด้วยตัวแปร vec3 ที่ให้ฟังก์ชัน textureCube ไปตัดกับกล่องลูกบาศก์ที่มีความยาวด้านละสองหน่วยที่มีจุดศูนย์กลางอยู่ที่จุด (0,0,0)

การใช้ Cube Map ใน GLSL



การทำกระจกใน Cg

- ให้ vertex program คำนวณ
  - ตำแหน่งใน world space ของแต่ fragment
  - Normal ใน world space แต่ละ fragment
- ให้ fragment program รับตำแหน่งของตา
- แล้วให้ fragment program คำนวณ
  - เวกเตอร์ทิศทางการสะท้อนแสงซึ่งเกิดจากแสงจากตา เดินทางไปยังตำแหน่งของ fragment
  - เอาเวกเตอร์ทิศทางที่ได้ไปอ่านสีจาก cube map

Vertex Program สำหรับทำกระจก

```
varying vec3 normal;
varying vec3 position;

void main()
{
    gl_Position = ftransform();
    position = (gl_ModelViewMatrix * gl_Vertex).xyz;
    normal = (gl_NormalMatrix * gl_Normal).xyz;
}
```

Vertex Program สำหรับทำกระจก

- ความหมายของตัวแปร
  - position ใช้เก็บตำแหน่งใน world space
  - normal ใช้เก็บเวกเตอร์ตั้งฉากใน world space

- เวลาเขียนโปรแกรมภาษา C ต้องทำให้ modelview matrix เป็น model เฉยๆ ซึ่งทำได้โดยการยก view matrix ไปคูณเข้ากับ projection matrix (เหมือนในคาบที่แล้ว)

Fragment Program สำหรับทำกระจก

```
uniform vec3 eye_position;
uniform samplerCube env;

varying vec3 position;
varying vec3 normal;

void main()
{
    vec3 n = normalize(normal);
    vec3 eye_to_point =
            normalize(position - eye_position);
    vec3 reflected = reflect(eye_to_point, n);
    gl_FragColor = textureCube(env, reflected);
}
```

Fragment Program สำหรับทำกระจก

- เราทำการคำนวณเวกเตอร์ทิศทางของแสงที่เดินทางจากตาไปยังตำแหน่งของ fragment ด้วยคำสั่ง

```
vec3 eye_to_point =
        normalize(position - eye_position);
```

- หลังจากนั้นคำนวณทิศทางที่แสงสะท้อนออกไปด้วยคำสั่ง

```
vec3 reflected = reflect(eye_to_point, n);
```

- ฟังก์ชัน reflect มีไว้สำหรับคำนวณเวกเตอร์แสงสะท้อน

Fragment Program สำหรับทำกระจก

- ขั้นสุดท้าย เรานำเอาทิศทางของเวกเตอร์แสงสะท้อนไปอ่านค่าจาก cube map

```
gl_FragColor = textureCube(env, reflected);
```

ดู demo

การทำวัตถุผิวมันวาว

- เราอาจมองว่าพื้นผิวของวัตถุมันวาวมีส่วนประกอบอยู่สองส่วน
  - ส่วนหนึ่งมีพฤติกรรมเหมือนกระจก
  - อีกส่วนมีพฤติกรรมตาม lighting model อื่น เช่น phong lighting model

- สีของพื้นผิวประเภทนี้เกิดจากการนำเอาสีที่ได้จากการสะท้อนแสงแบบกระจกมารวมกับสีที่ได้จากพฤติกรรมการสะท้อนแสงอื่นๆ

การทำวัตถุผิวมันวาว

- เราสามารถคำนวณสีของทั้งสองส่วน
  - สมมติว่าส่วนแรกได้สี a และ
  - ส่วนที่สองได้สี b

- เรากำหนดเลข w (ย่อคำว่า weight) โดยที่ $0 \leq w \leq 1$ แล้วให้สีขั้นสุดท้ายมีค่าเท่ากับ

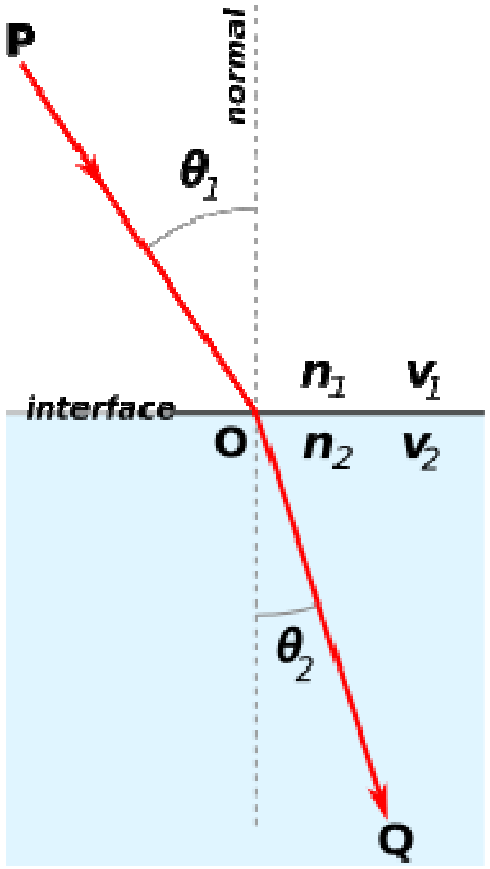
$$color = w \times a + (1-w) \times b$$

ดู demo

การหักเหแสง

การจำลองการหักเหของแสง

- การหักเหของแสงเป็นไปตาม **กฎของสเนลล์** (Snell's Law)



$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

เมื่อ

- η_1 คือดรรชนีหักเหของตัวกลางแรก
- η_2 คือดรรชนีหักเหของตัวกลางที่สอง
- θ_1 และ θ_2 คือมุมระหว่างทางเดินของแสงกับ normal ตามรูป

การจำลองการหักเหของแสง

- ค่าดรรชนีหักเหของแสงเป็นค่าคงตัวของตัวกลางแบบต่างๆ
  - ถ้าต้องการจะรู้ค่าก็ต้องไปเปิดหนังสือ
- ภาษา GLSL มีฟังก์ชันอำนวยความสะดวกให้เราสามารถคำนวณทิศทางในการหักแสงได้อย่างง่ายดาย คือ ฟังก์ชัน reflect

  refract(vec3 I, vec3 N, float etaRatio)
  - I คือเวกเตอร์ที่แสงเดินทางเข้าวัตถุ (เวกเตอร์ PO)
  - N คือเวกเตอร์ตั้งฉาก
  - etaRatio คือค่า η_1 / η_2
  - ฟังก์ชันนี้จะคืนค่าทิศทางที่แสดงหักเหออกมา (เวกเตอร์ OQ)

การสร้างแก้วใน GLSL

- ใช้หลักการเดียวกับการสร้างกระจกในคาบที่แล้ว
  - ใช้ fragment program คำนวณทิศทางที่แสงหักเห
  - นำทิศทางที่แสงหักเหไปอ่านค่าจาก cube map แล้วนำค่านี้มาเป็นสี
- เราทำการคำนวณทุกอย่างใน world space
  - ดังนั้นใช้ vertex program ตัวเดียวกับโปรแกรมที่แล้วได้

Fragment Program สำหรับจำลองแก้ว

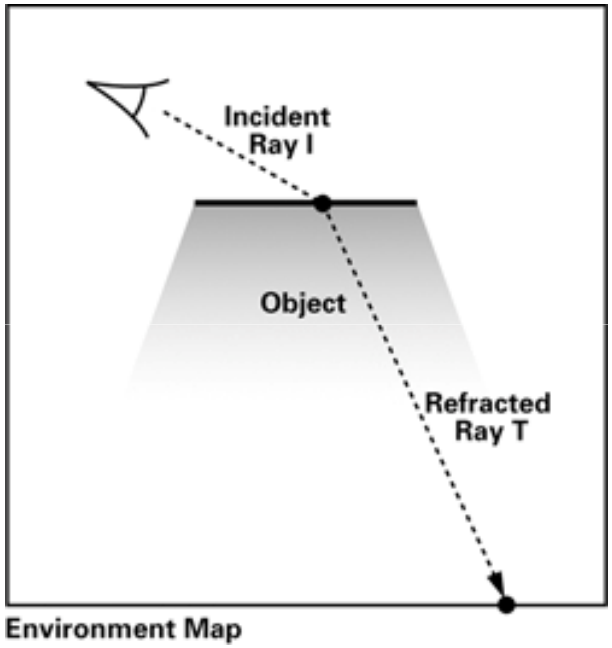
```
uniform vec3 eye;
uniform samplerCube env;
uniform float eta_ratio;

varying vec3 position;
varying vec3 normal;

void main()
{
    vec3 n = normalize(normal);
    vec3 eye_to_point = normalize(position - eye);
    vec3 refracted =
        refract(eye_to_point, n, eta_ratio);
    vec4 refraction = textureCube(env, refracted);

    gl_FragColor = refraction;
}
```

ดู demo

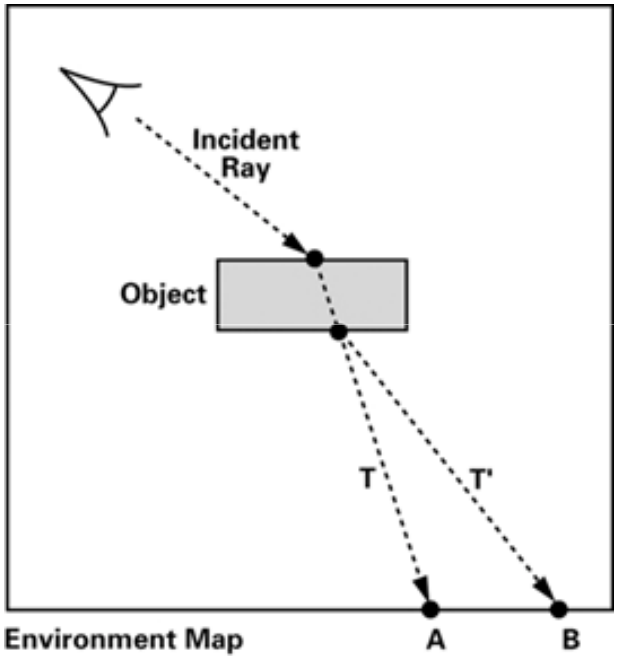
ข้อสังเกต

- การหักเหแสงที่เห็นใน demo ไม่ใช้การหักเหแสงแบบที่เกิดขึ้นจริงในธรรมชาติ
- ทำไม? เพราะมันมีการหักเหแสงแค่ครั้งเดียว! ความจริงรูปโดนัทเป็นรูปปิด เพราะฉะนั้นกว่าแสงจะไปหา skybox ได้ต้องมีการหักเหมากกว่าหนึ่งครั้ง

ข้อสังเกต (ต่อ)



สิ่งที่เราทำ



สิ่งที่ควรจะเป็น

ข้อสังเกต (ต่อ)

- ทำไมเราถึงไม่ทำให้มันตรงกับความเป็นจริง?
- เพราะมันทำยากมาก!
  - เราต้องรู้ว่าแสงเข้าไปสู่วัตถุที่จุดไหน
  - เราต้องรู้ว่าแสงออกไปสู่วัตถุที่จุดไหน
  - OpenGL หรือ GLSL ไม่ได้เก็บข้อมูลเหล่านี้ไว้ให้เรา
- ดังนั้นเราจึง “โกง” ด้วยการคิดว่ามีการหักเหแสงแค่ครั้งเดียว
- ในคอมพิวเตอร์กราฟิกส์ การโกงเช่นนี้เป็นเรื่องที่พบเห็นได้บ่อยมาก เพราะมันช่วยทำให้ภาพสวยได้โดยไม่ต้องเปลืองเวลาคำนวณมาก

ปรากฏการณ์ Fresnel

- จะเห็นว่าตัวอย่างที่ demo ไปยังไม่ค่อยสวยและเหมือนจริงเท่าไหร่
- เพราะแก้วนั้นไม่ได้หักเหแสงเพียงอย่างเดียว มันยังมีการสะท้อนแสงด้วย
- ปรากฏการณ์ที่แสงส่วนสะท้อนที่พื้นผิววัตถุแต่แสงบางส่วนหักเหที่พื้นผิววัตถุเรียกว่า **ปรากฏการณ์ Fresnel**
- การคำนวณปริมาณแสงที่สะท้อนและหักเหจริงๆ ในธรรมชาตินั้นใช้สูตรที่ซับซ้อน ดังนั้นเราจะโกงโดยฝันสูตรขึ้นมาเอง

ปรากฏการณ์ Fresnel (ต่อ)

- สมมติว่าเราคำนวณสีที่ได้จากการสะท้อนแสงเก็บไว้ในตัวแปร a
- และคำนวณสีที่ได้จากการหักเหแสงไว้ในตัวแปร b
- เราต้องการเอาสีทั้งสองมารวมกัน เพื่อจำลองปรากฏการณ์ Fresnel ด้วยสูตร:

$$color = w \times a + (1-w) \times b$$

ปรากฏการณ์ Fresnel (ต่อ)

- เราสามารถคำนวณค่า **w** ได้ดังต่อไปนี้ (สูตรนี้ฝันขึ้นมาเอง)

$$w = \max(0, \min(1, bias + scale \times (1 + (I \cdot N)^{power})))$$

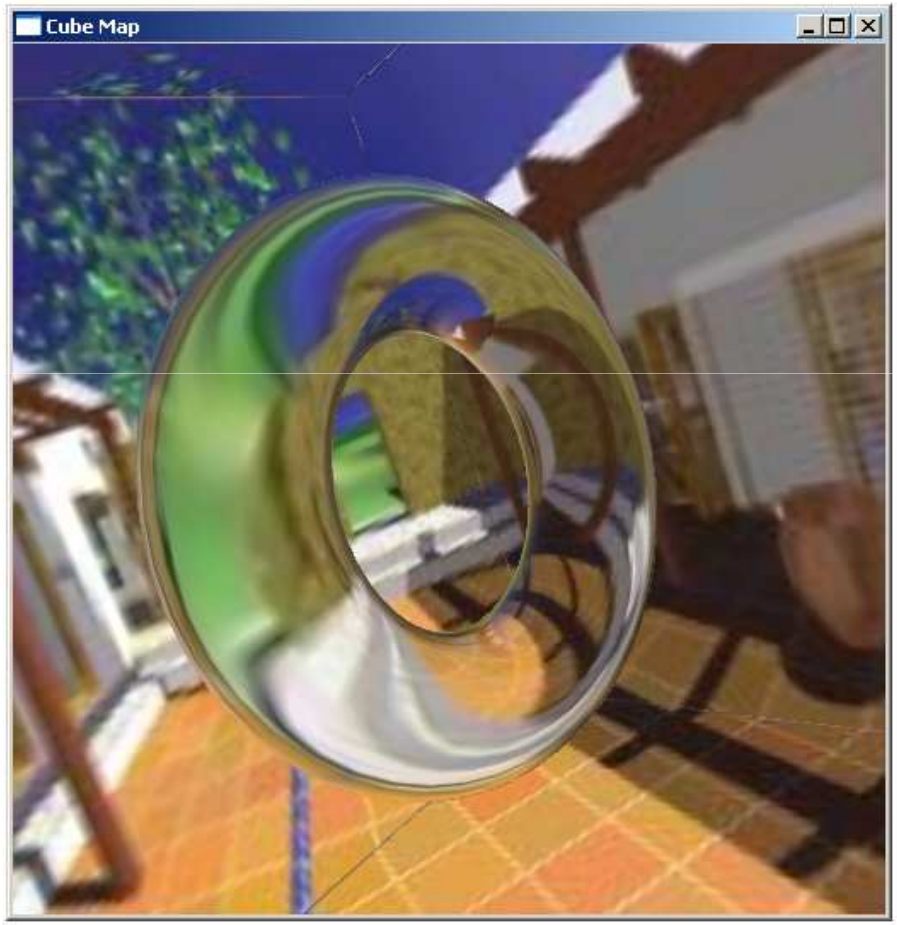
โดยที่

- bias, scale, และ power คือค่าที่ผู้ใช้กำหนดขึ้นมา
- I คือเวกเตอร์ทิศทางที่แสงเดินทางเข้าสู่วัตถุ
- N คือเวกเตอร์ตั้งฉาก

Fragment Program
สำหรับจำลองปรากฏการณ์ Fresnel

```
uniform vec3 eye;
uniform samplerCube env;
uniform float eta_ratio;
uniform float fresnel_bias;
uniform float fresnel_scale;
uniform float fresnel_power;

varying vec3 position;
varying vec3 normal;

void main()
{
    vec3 n = normalize(normal);
    vec3 eye_to_point = normalize(position - eye);
    vec3 refracted = refract(eye_to_point, n, eta_ratio);
    vec3 reflected = reflect(eye_to_point, n);
    vec4 refraction = textureCube(env, refracted);
    vec4 reflection = textureCube(env, reflected);

    float reflected_weight = fresnel_bias +
        fresnel_scale * pow(1.0 + dot(n, eye_to_point), fresnel_power);
    reflected_weight = max(0.0, min(reflected_weight, 1.0));

    gl_FragColor = reflected_weight * reflection + (1.0-reflected_weight) * refraction;
}
```

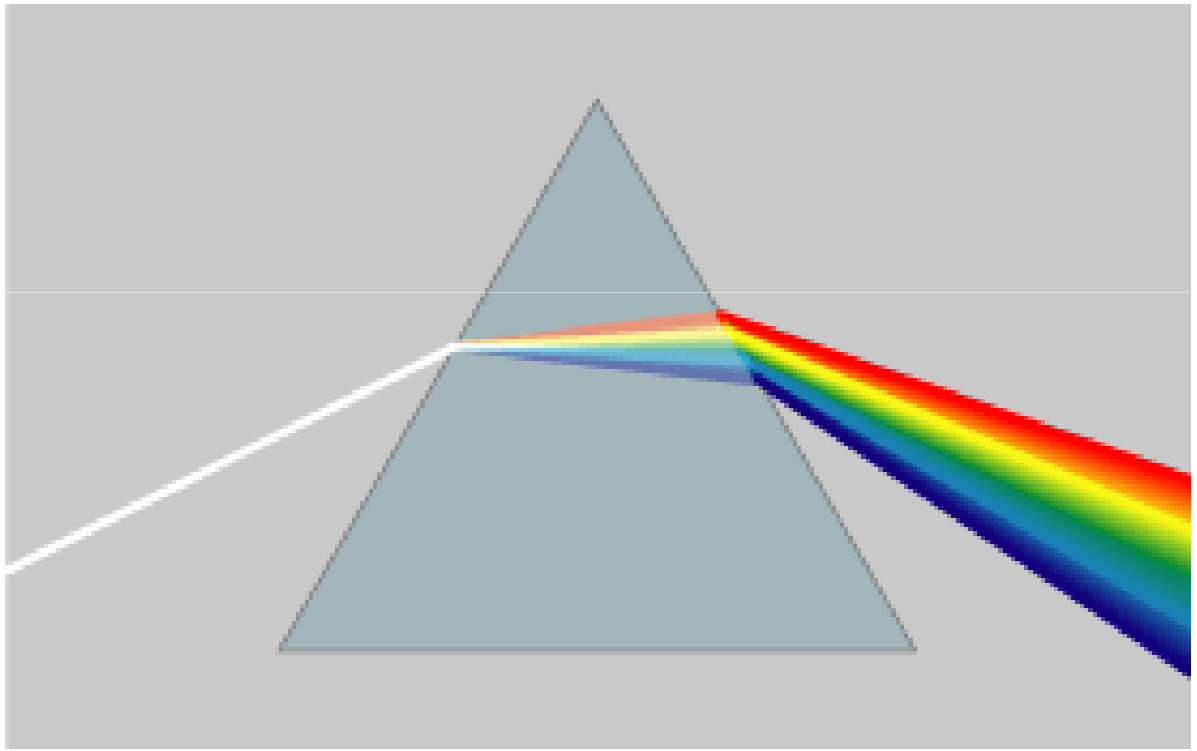
ดู demo



Chromatic Dispersion

- แสงขาวที่เราเห็น ความจริงประกอบด้วยแสงความถี่ต่างๆ สีต่างๆ มากมาย
- แสงสีต่างๆ เหล่านี้เมื่อหักเหจะมีมุมในการหักเหต่างๆ กัน ทำให้แยกออกจากกันมาเป็นแถบๆ
- เราจึงเห็นปรากฏการณ์เช่น รุ้งกินน้ำ หรือแถบสีเมื่อแสงเดินทางผ่านปริซึม

Chromatic Dispersion (ต่อ)



การจำลอง Chromatic Dispersion ใน Cg

- สีใน OpenGL มีสามย่านความถี่คือ R, G, และ B
- เราสามารถแค่กำหนดให้ eta_ratio ของสามย่านความถี่นี้มีค่าแตกต่างกัน เพื่อจำลอง chromatic dispersion ได้

Fragment Program
สำหรับจำลอง Chromatic Dispersion

```
uniform vec3 eye;
uniform samplerCube env;
uniform vec3 eta_ratio;
uniform float fresnel_bias;
uniform float fresnel_scale;
uniform float fresnel_power;

varying vec3 position;
varying vec3 normal;

void main()
{
    vec3 n = normalize(normal);
    vec3 eye_to_point = normalize(position - eye);

    vec3 refracted_r = refract(eye_to_point, n, eta_ratio.r);
    vec3 refracted_g = refract(eye_to_point, n, eta_ratio.g);
    vec3 refracted_b = refract(eye_to_point, n, eta_ratio.b);

    vec3 refraction_r = textureCube(env, refracted_r).rgb * vec3(1,0,0);
    vec3 refraction_g = textureCube(env, refracted_g).rgb * vec3(0,1,0);
    vec3 refraction_b = textureCube(env, refracted_b).rgb * vec3(0,0,1);

    vec4 refraction = vec4(refraction_r + refraction_g + refraction_b, 1);
```

Fragment Program
สำหรับจำลอง Chromatic Dispersion

```
    vec3 reflected = reflect(eye_to_point, n);
    vec4 reflection = textureCube(env, reflected);

    float reflected_weight = fresnel_bias + fresnel_scale * pow(1.0 + dot(n, eye_to_point),
    fresnel_power);
    reflected_weight = max(0.0, min(reflected_weight, 1.0));

    gl_FragColor = reflected_weight * reflection + (1.0-reflected_weight) * refraction;
}
```

ดู demo