CS559: Computer Graphics

Lecture 27: Texture Mapping

Li Zhang

Spring 2008

Many slides from Ravi Ramamoorthi, Columbia Univ, Greg Humphreys, UVA and Rosalee Wolfe, DePaul tutorial teaching texture mapping visually, Jingyi Yu, U Kentucky.

Today

- Finish Texture mapping, continue on Blending and Shape Modeling

- Reading
  - Redbook: Ch 9, Ch 6 (Blending only)
  - (highly recommended) Moller and Haines: *Real-Time Rendering, 3e*, Ch 6
    - Linux: /p/course/cs559-lizhang/public/readings/6_texture.pdf
    - Windows: P:\course\cs559-lizhang\public\readings\6_texture.pdf
  - (optional) Shirley: Ch 11.4 – 11.8

Shadow map

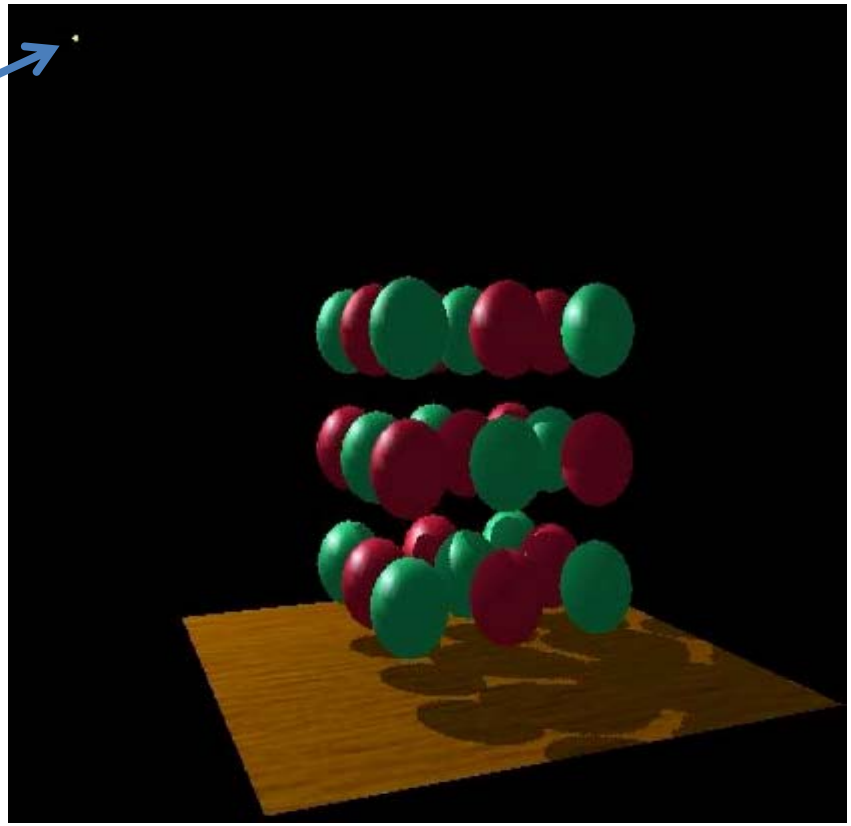
- Cast shadow on curved/non planar surfaces

Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78

Shadow map

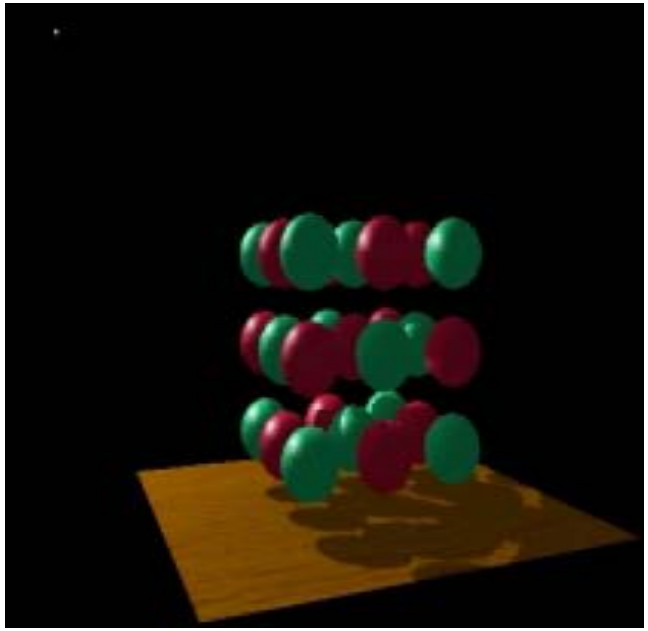
Let’s consider this scene:

Point light source

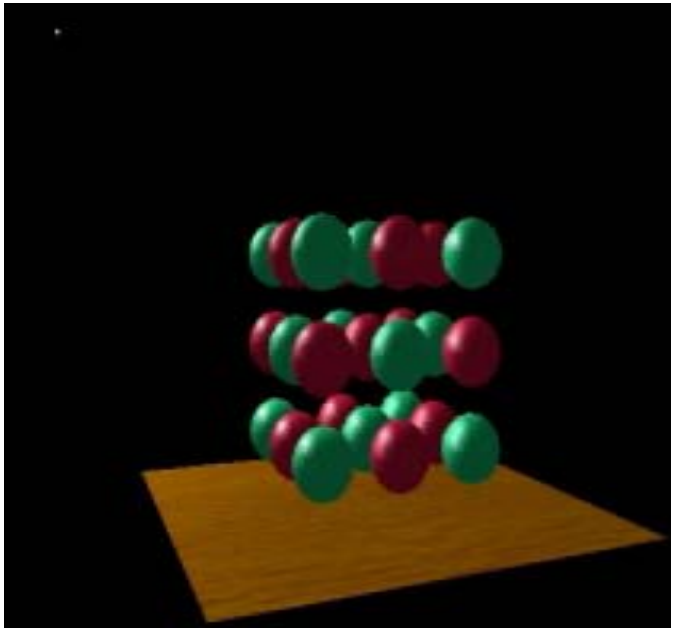


Q: Is hack shadow enough?

Shadow map

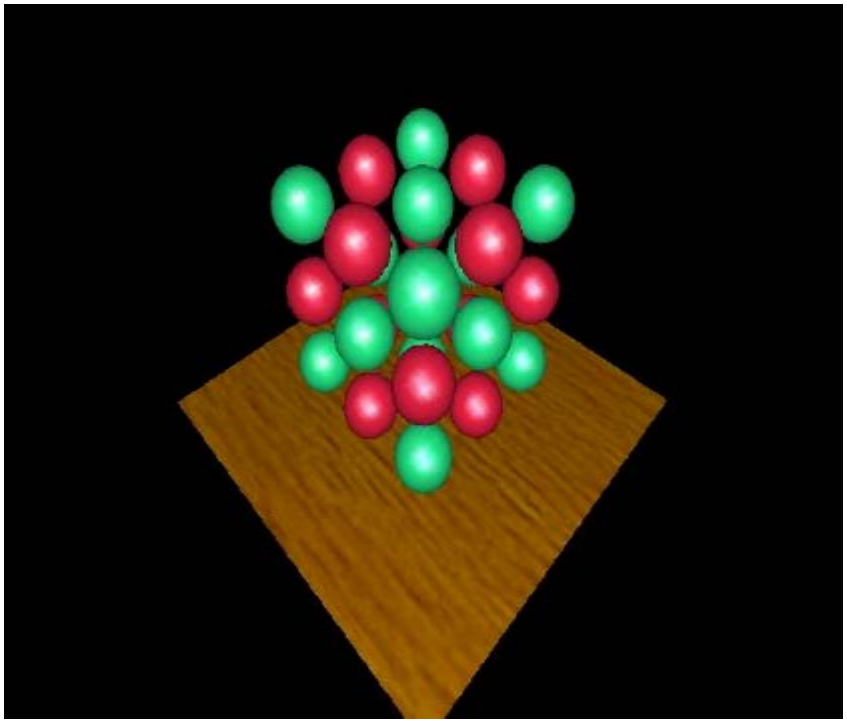


With shadow



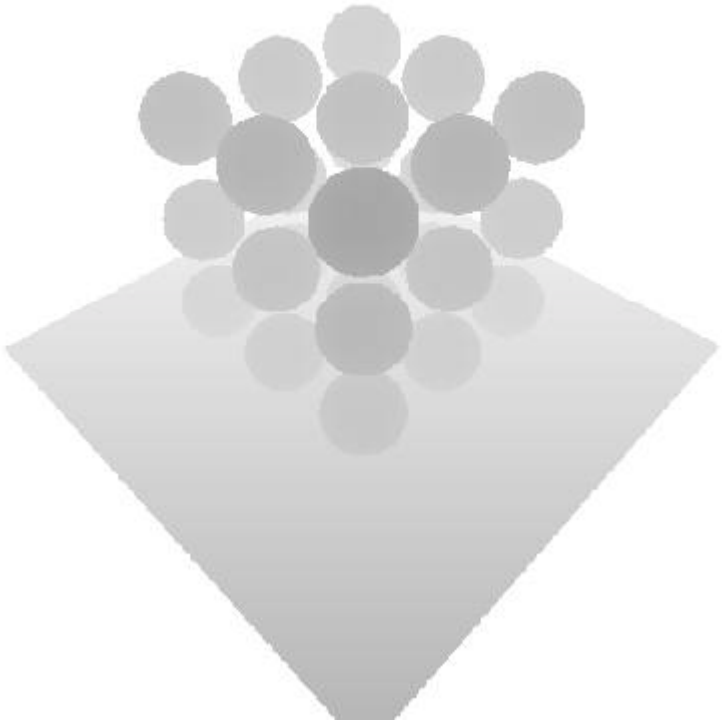
Without shadow

Shadow map



Scene from the light’s viewpoint

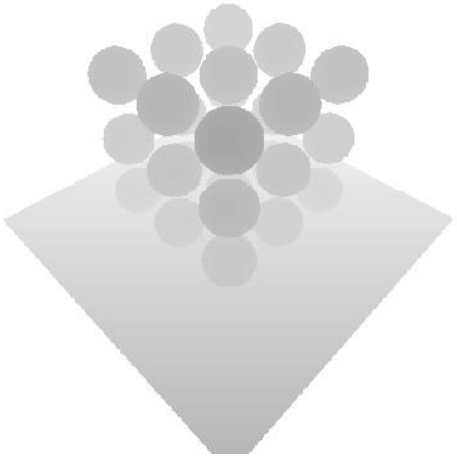
Shadow map



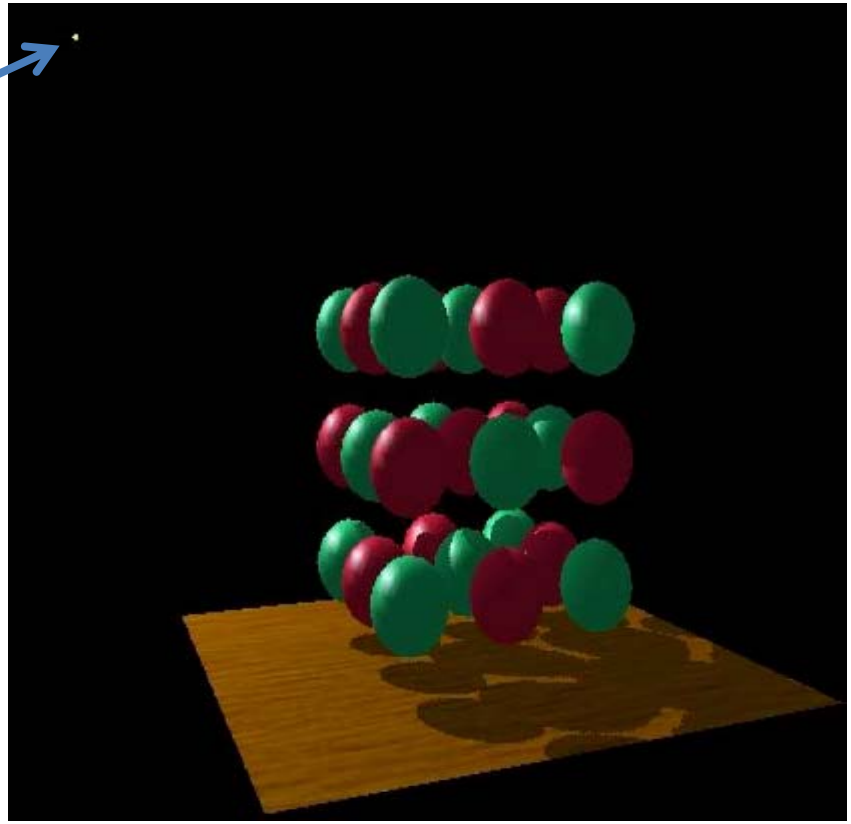
Depth map from the light’s viewpoint

Shadow map

Point light source

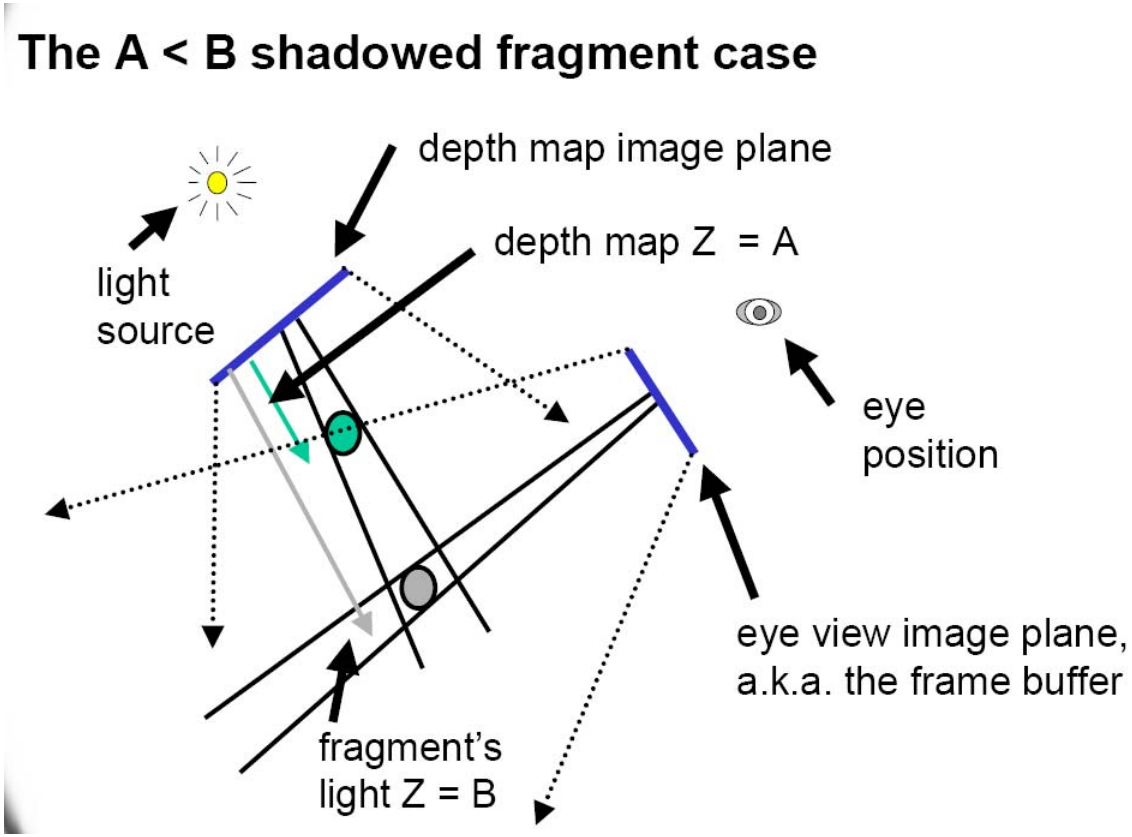


Depth Texture



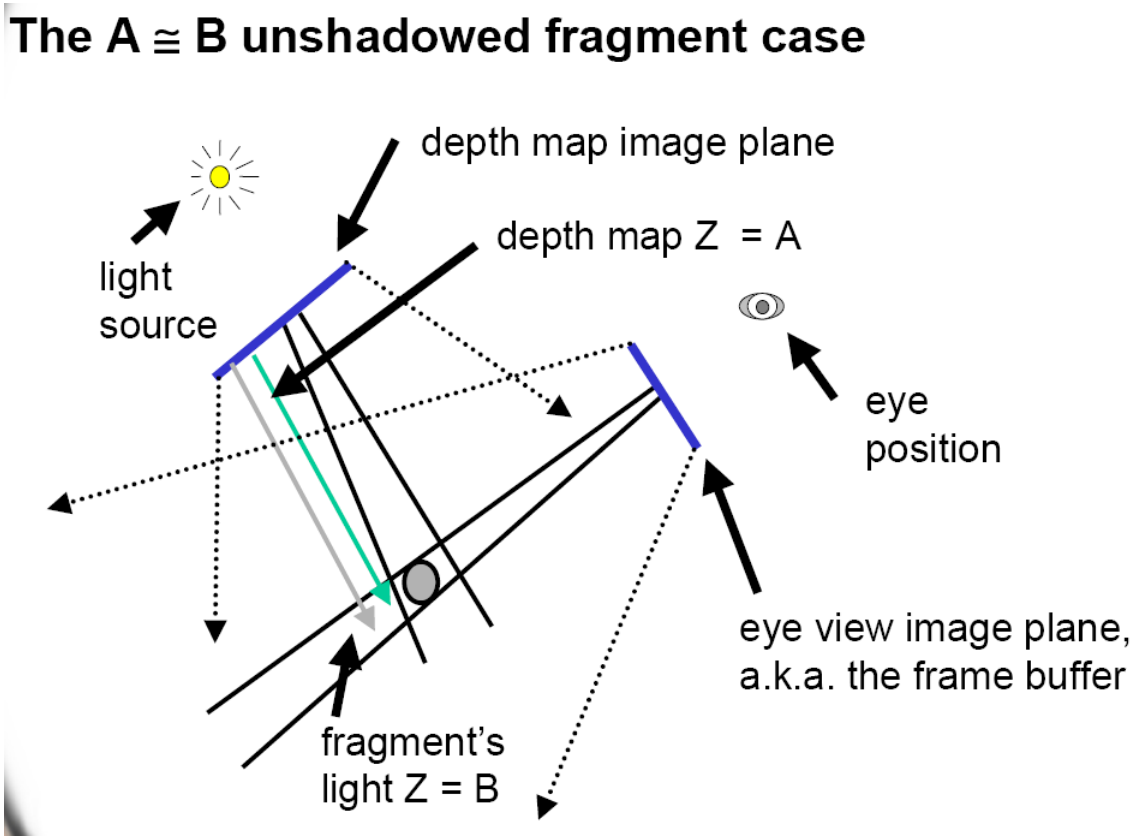
In rasterization, check light visibility using the depth map with respect to the light.

Shadow map




```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
GL_COMPARE_R_TO_TEXTURE);
```

Shadow map




```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
GL_COMPARE_R_TO_TEXTURE);
```

Shadow Map

- Depth testing from the light's point-of-view

A Two pass algorithm:

First, render depth buffer from the light's point-of-view

- the result is a "depth map" or "shadow map" essentially a 2D function indicating the depth of the closest pixels to the light
- This depth map is used in the second pass

Shadow Map

- Shadow determination with the depth map

  Second, render scene from the eye's point-of-view
  - For each rasterized fragment
    - determine fragment's XYZ position relative to the light
    - compare the depth value for XYZ and the depth value in the buffer computed in the first pass, relative to the light.

More details, see Red book v2.1. pp459-463
Free Version 1.1 does not have this feature.
Also see a tutorial
http://www.paulsprojects.net/tutorials/smt/smt.html

Project 3

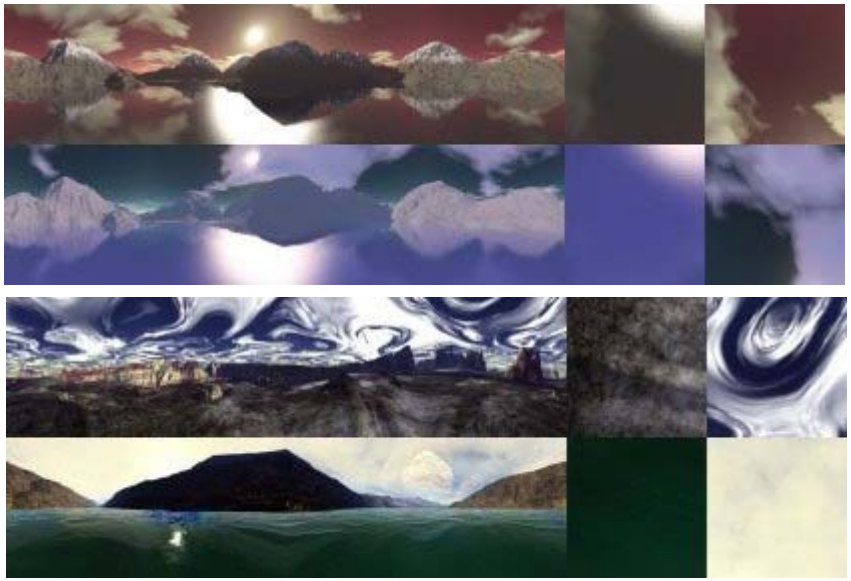
- Technical Challenge 1: sky box/sky dome
  - Render the environment within a box or hemisphere with sky texture
  - Google “sky box texture”, you can get many images for this.




http://mpan3.homeip.net/earth

Project 3

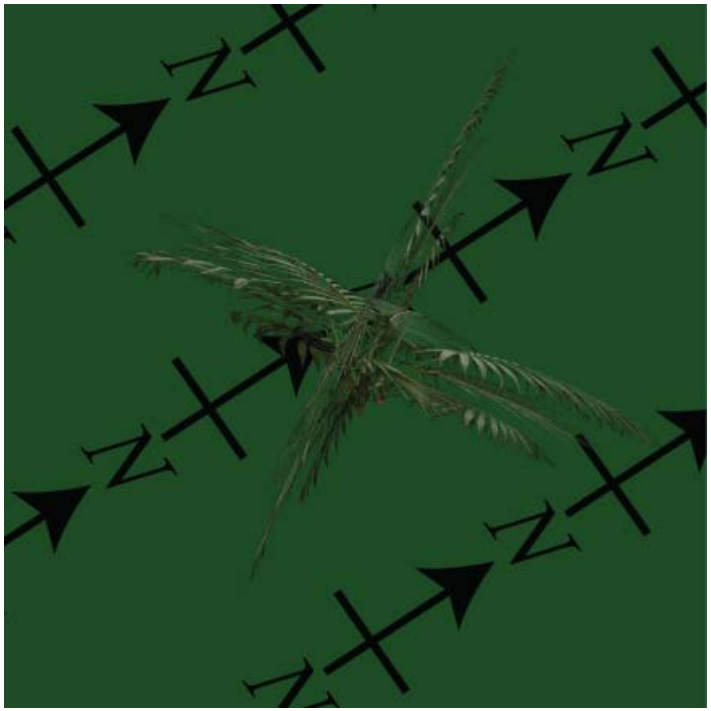
- Technical Challenge: sky box/sky dome
  - Render the environment within a box or hemisphere with sky texture
  - Google “sky box texture”, you can get many images for this.



http://skymatter.thegamecreators.com/?f=pack1

Cross-tree using alpha

Side view



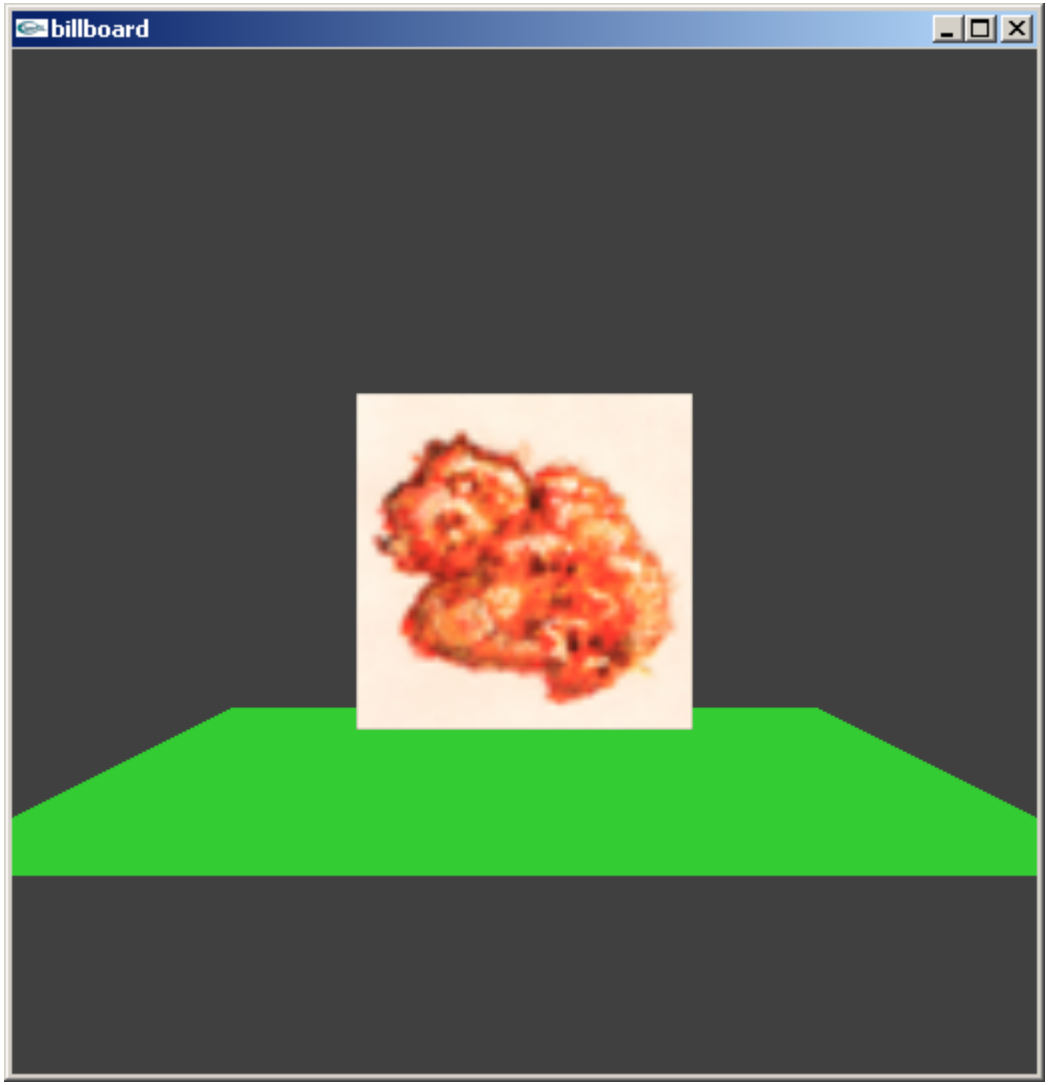
Top View

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.05);
```

Recap on Texture mapping

- Computing Perspective Correct Texture Coord
- Sampling Texture Values (Mipmap, …)
- OpenGL single texture mapping
- Projector Texture Mapping
- Bump Map
- Displacement Map
- 3D Texture
- Environment Map
- Multi-texturing
- Shadow map

Blending



Billboard

Blending in OpenGL

Source --- In coming fragment color: (Rs, Gs, Bs, As)

*

Modulate Source color by (Sr, Sg, Sb, Sa)

+

Destination --- Current Frame buffer color: (Rd, Gd, Bd, Ad)

*

Modulate Destination color by (Dr, Dg, Db, Da)

Final Color: (?, ? , ?, ?)

Example:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Blending in OpenGL

Source --- In coming fragment color: (Rs, Gs, Bs, As)

*

Modulate Source color by (Sr, Sg, Sb, Sa)

+

Destination --- Current Frame buffer color: (Rd, Gd, Bd, Ad)

*

Modulate Destination color by (Dr, Dg, Db, Da)

Final Color: (?, ? , ?, ?)

Example:

```
glEnable(GL_BLEND);
glBlendColor(0.3,0.4,0.5,0.6);
glBlendFunc(GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR);
```

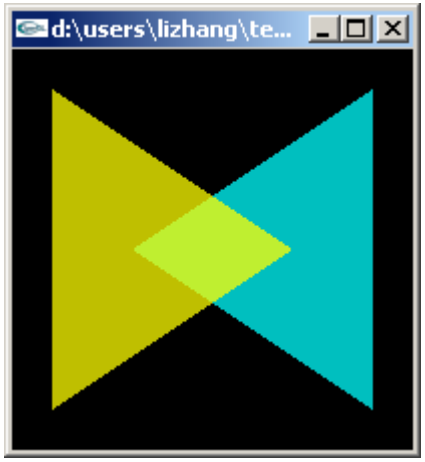
Other choices

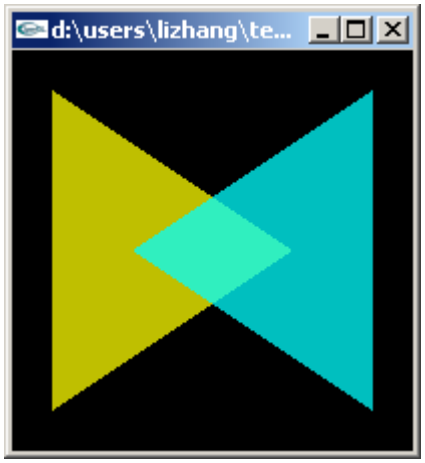
Table 6-1 : Source and Destination Blending Factors

Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	destination	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	source	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	source or destination	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPHA	source or destination	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	source or destination	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPHA	source or destination	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

Red Book ch 6

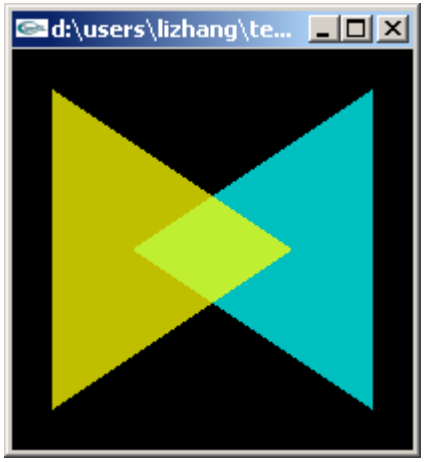
Blending depends on Order

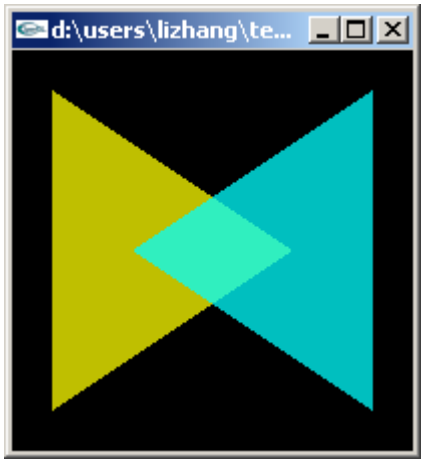




- In 3D, the problem is tricky
  - If an opaque obj is in front of a translucent obj
    - Draw opaque
  - If a translucent obj is in front of an opaque
    - Blend

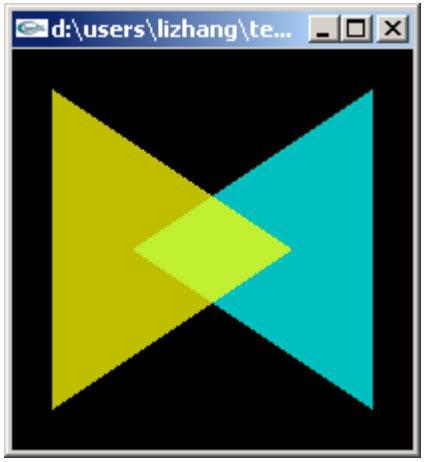
Blending depends on Order

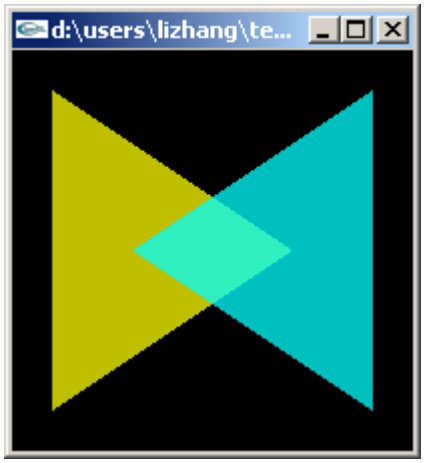




- Solution
  - A-buffer
  - BSP tree

Blending depends on Order





- Hack
  - Draw opaque ones first
  - Freeze depth map, glDepthMask(GL_FALSE);
  - Draw transparent ones
    - If behind the depth map, hide
    - If in front of depth map, blend
- When will it introduce artifacts?