Shadows

Thanks to:
Frédéric Durand
and Seth Teller
MIT

Shadows

• Realism
• Depth cue

Shadows as depth cue

Spatial relationship between objects

Michael McCool
Univ of Waterloo

Spatial relationship between objects

Spatial relationship between objects
Spatial relationship between objects

Shadows add visual acuity
- Shadows as the origin of painting

Shadows and art
- Only in Western pictures (here Caravaggio)

Duality shadow-view
- A point is lit if it is visible from the light source
- Shadow computation very similar to view computation
Shadow ray
- Ray from visible point to light source
- If blocked, discard light contribution
- One shadow ray per light
- Optimization?
  - Stop after first intersection (don’t worry about tmin)
  - Test latest obstacle first

Ray-casting shadows

Overview
- Project Shadows
- Shadow map
  - Image-precision, texture mapping
- Shadow volume
  - Object space
- Soft shadows

Planar Projection
- Render a ground-plane
- Render an object
- Then render the object again, but this time
  - Projected onto the plane
  - Without light, so that the shadow is black
  - Half-transparent (using blending), to avoid completely dark shadows
  - Avoid multiple “darkening” on one spot by using ordinary z-buffer checks

Planar Shadows (& demo)

Shadow is projected into the plane of the floor.

Constructing a Shadow Matrix

```c
void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4])
{
    GLfloat dot;
    /* Find dot product between light position vector and ground plane normal. */
    dot = groundplane[0] * lightpos[0] +
         groundplane[1] * lightpos[1] +
         groundplane[3] * lightpos[3];
    shadowMat[0][0] = dot - lightpos[0] * groundplane[0];
    shadowMat[1][0] = 0.f - lightpos[0] * groundplane[1];
    shadowMat[2][0] = 0.f - lightpos[0] * groundplane[2];
    shadowMat[3][0] = 0.f - lightpos[0] * groundplane[3];
    shadowMat[0][1] = 0.f - lightpos[1] * groundplane[0];
    shadowMat[1][1] = dot - lightpos[1] * groundplane[1];
    shadowMat[2][1] = 0.f - lightpos[1] * groundplane[2];
    shadowMat[3][1] = 0.f - lightpos[1] * groundplane[3];
    shadowMat[0][2] = 0.f - lightpos[2] * groundplane[0];
    shadowMat[1][2] = 0.f - lightpos[2] * groundplane[1];
    shadowMat[3][2] = 0.f - lightpos[2] * groundplane[3];
    shadowMat[0][3] = 0.f - lightpos[3] * groundplane[0];
    shadowMat[1][3] = 0.f - lightpos[3] * groundplane[1];
}
```
How to Render the Shadow

```c
/* Render 50% black shadow color on top of whatever the floor appearance is. */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);
disable(GL_LIGHTING); /* Force the 50% black. */
gColor4f(0.0, 0.0, 0.0, 0.5);

/* Project the shadow. */
gMultMatrixf((GLfloat *) floorShadow);
drawDinosaur();

/* Project the floor. */
gMultMatrixf((GLfloat *) floorShadow);
drawDinosaur();

/* Project the floor again. */
gMultMatrixf((GLfloat *) floorShadow);
drawDinosaur();
```

Not Quite So Easy (1)

Without stencil to avoid double blending of the shadow pixels:

```
Notice dark spots on the planar shadow.
```

Solution: Clear stencil to zero. Draw floor with stencil of one. Draw shadow if stencil is one. If shadow’s stencil test passes, set stencil to two. No double blending.

Not Quite So Easy (2)

There’s still another problem even if using stencil to avoid double blending.

```
Shadow fights with depth values from the floor plane. Use polygon offset to raise shadow polygons slightly in Z.
```

Not Quite so Easy (3)

```
Good.
Notice right image’s reflection falls off the floor!
Same problem with Shadows!
```

Planar Projection

- Fast
  - Can be done with a matrix operation
- Easy
  - Just use the Modelview transform
- Very unrealistic
  - Just planar shadows

Shadow Quality: “Blobs”
Fake shadows using textures

- Separate obstacle and receiver
- Compute b/w image of obstacle from light
- Use textures, (s,t) for each receiver

Image from light source  BW image of obstacle  Final image

Figure from Moller & Haines “Real Time Rendering”

Fake shadows using textures

- Limitations?

Image from light source  BW image of obstacle  Final image

Figure from Moller & Haines “Real Time Rendering”

Shadow maps

- Use texture mapping but using depth
- 2 passes (at least)
  - Compute shadow map from light source
    - Store depth buffer (shadow map)
  - Compute final image
    - Look up the shadow map to know if points are in shadow

Figure from Foley et al. “Computer Graphics Principles and Practice”

Shadow map look up

- We have a 3D point x,y,z
- How do we look up the shadow map?

Figure from Foley et al. “Computer Graphics Principles and Practice”

Shadow maps

- In Renderman
  - (High-end production software)
Shadow maps

• Can be done in hardware (Segal '92 paper)
• Using hardware texture mapping
  – Texture coordinates u,v,w generated using 4x4 matrix
  – Modern hardware permits tests on texture values

Introducing Another Technique: Shadow Mapping

• Image-space shadow determination
  – Lance Williams published the basic idea in 1978
  – By coincidence, same year Jim Blinn invented bump mapping
    (a great vintage year for graphics)
  – Completely image-space algorithm
    • means no knowledge of scene’s geometry is required
    • must deal with aliasing artifacts
  – Well known software rendering technique
    • Pixar’s RenderMan uses the algorithm
    • Basic shadowing technique for Toy Story, etc.

Shadow Mapping References

• Important SIGGRAPH papers
  – Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78
  – William Reeves, David Salesin, and Robert Cook ( Pixar), “Rendering antialiased shadows with depth maps,” SIGGRAPH 87

The Shadow Mapping Concept (3)

• The Shadow Map Comparison
  – Two values
    • A = Z value from depth map at fragment’s light XY position
    • B = Z value of fragment’s XYZ light position
  – If B is greater than A, then there must be something closer to the light than the fragment
    • then the fragment is shadowed
  – If A and B are approximately equal, the fragment is lit

Shadow Mapping with a Picture in 2D (1)

The A < B shadowed fragment case

Light source
depth map image plane
depth map Z' = A

Eye position
Fragment’s light Z = B

Eye view image plane, a.k.a. the frame buffer

Shadow Mapping with a Picture in 2D (2)

The A \cong B unshadowed fragment case

Light source
depth map image plane
depth map Z' = A

Eye position
Fragment’s light Z = B

Eye view image plane, a.k.a. the frame buffer
The depth map could be at a different resolution from the framebuffer. This mismatch can lead to artifacts.

Visualizing the Shadow Mapping Technique (1)
- A fairly complex scene with shadows

Visualizing the Shadow Mapping Technique (2)
- Compare with and without shadows

Visualizing the Shadow Mapping Technique (3)
- The scene from the light’s point-of-view

Visualizing the Shadow Mapping Technique (4)
- The depth buffer from the light’s point-of-view

Visualizing the Shadow Mapping Technique (5)
- Projecting the depth map onto the eye’s view
Visualizing the Shadow Mapping Technique (6)

- Projecting light’s planar distance onto eye’s view

Visualizing the Shadow Mapping Technique (6)

- Comparing light distance to light depth map

Visualizing the Shadow Mapping Technique (7)

- Scene with shadows

Notice how specular highlights never appear in shadows

Notice how curved surfaces cast shadows on each other

Problems with shadow maps?

- Field of view
- Bias
- Aliasing

Field of view problem

- What if point to shadow is outside field of view of shadow map?
- Use cubical shadow map
- Use only spot lights!

Problems with shadow maps?

- Field of view
- Bias
- Aliasing
The bias nightmare

• For a point visible from the light source
  ShadowMap(x’,y’) = z’
• Avoid erroneous self shadowing

Bias for shadow maps

ShadowMap(x’,y’)+bias < z’
Choosing the good bias value can be very tricky

Construct Light View Depth Map

• Realizing the theory in practice
  – Constructing the depth map
    • use existing hardware depth buffer
    • use glPolygonOffset to bias depth value
    • read back the depth buffer contents
  – Depth map can be copied to a 2D texture
    • unfortunately, depth values tend to require more precision than 8-bit typical for textures
    • depth precision typically 16-bit or 24-bit

Justification for glPolygonOffset When Constructing Shadow Maps

• Depth buffer contains “window space” depth values
  – Post-perspective divide means non-linear distribution
  – glPolygonOffset is guaranteed to be a window space offset
• Doing a “clip space” glTranslatef is not sufficient
  – Common shadow mapping implementation mistake
  – Actual bias in depth buffer units will vary over the frustum
  – No way to account for slope of polygon

Depth Map Bias Issues

• How much polygon offset bias depends
  Too little bias, everything begins to shadow
  Too much bias, shadow starts too far back
  Just right

Selecting the Depth Map Bias

• Not that hard
  – Usually the following works well
    • glPolygonOffset(scale = 1.1, bias = 4.0)
  – Usually better to error on the side of too much bias
    • adjust to suit the shadow issues in your scene
  – Depends somewhat on shadow map precision
    • more precision requires less of a bias
  – When the shadow map is being magnified, a larger scale is often required
Problems with shadow maps?

- Field of view
- Bias
- Aliasing

Shadow map aliasing

- Undersampling of shadow map
- Reprojection aliasing

Alasing

- Finite shadow map resolution
- Result: pixelized shadows

Shadow map filtering (Reeves paper)

- Does not work!
- Filtering depth is not meaningful

Percentage closer filtering

- Filter the result of the test
- But makes the bias issue more tricky

Percentage closer filtering

- 5x5 samples
- Nice antialiased shadow
- Using a bigger filter produces fake soft shadows
- But makes the bias issue more tricky
Shadows in production

- Often use shadow maps
- Ray casting as fallback in case of robustness issues

Movie Time!

Aliasing

- Bad aliasing cases:
  - Large Scenes
    - High resolution shadow map required
  - Close-ups to shadow boundaries
    - Zoom in
  - Shadow stretches along the receiver

Aliasing

- Duelling frusta
  - Light shines opposite to viewing direction

Aliasing

- Duelling frusta
  - Resolution mismatch

Aliasing

- Miner’s headlamp
  - Similar frusta
  - Similar sampling
  - One shadowmap pixel for image pixel
Hardware Support

• OpenGL 1.4
  – GL_ARB_DEPTH_TEXTURE
    internal texture format to store depth values
  – glTexGen
    generation of light space coordinates as texture coordinates
  – GL_ARB_SHADOW
    special texture mode:
    \[ \text{return } \text{texture}(s, t) < r \ ? \ \text{Black} \ : \ \text{White}; \]

Hardware Support

• P-Buffers
  – offscreen rendering of shadow map
  – large shadow maps sizes

• GL_ARB_RENDER_TEXTURE and WGL_NV_RENDER_TEXTURE
  – bind P-Buffer depth buffer as depth texture
  – no copy needed

Pros and Cons

+ general
  – everything that can be rendered can cast and receive a shadow
  – works together with vertex programs

+ fast
  – full hardware support
  – (almost) no overhead for static scenes
  – two passes needed for dynamic scenes

+ robust
+ easy to implement
  - aliasing

Questions?