CSE 167: Introduction to Computer Graphics Lecture #17: Shadow Mapping

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Spring Quarter 2015

Announcements

- 3rd blog entry due this Sunday
- Final project presentations in CSE 1202 on Tuesday, June 9th

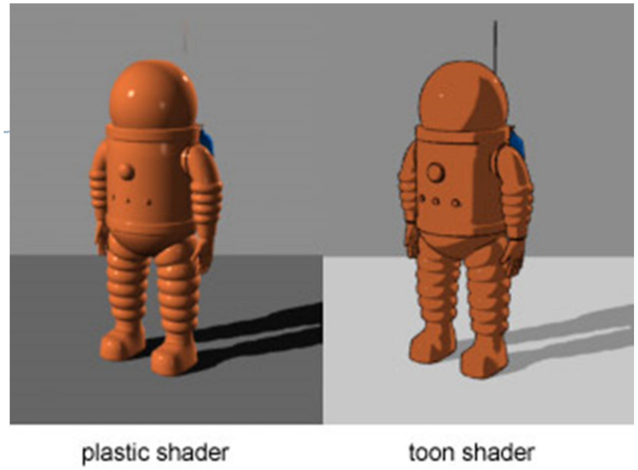
Lecture Overview

Advanced Shader Effects

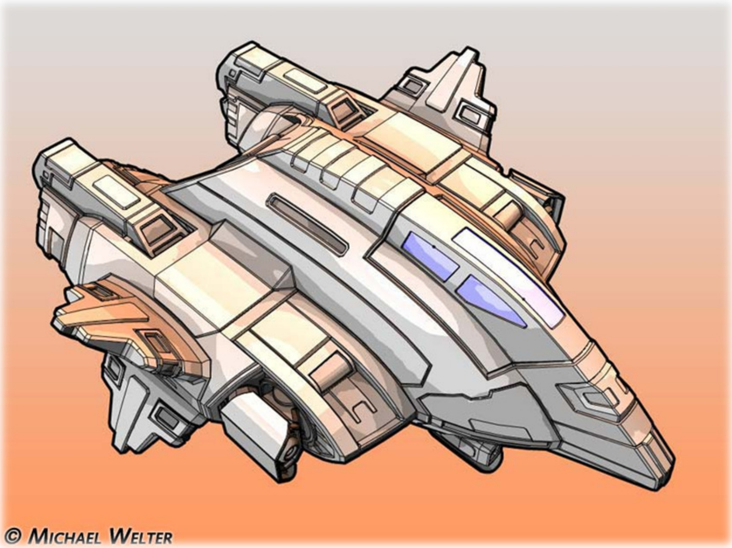
- Toon shading

Toon Shading

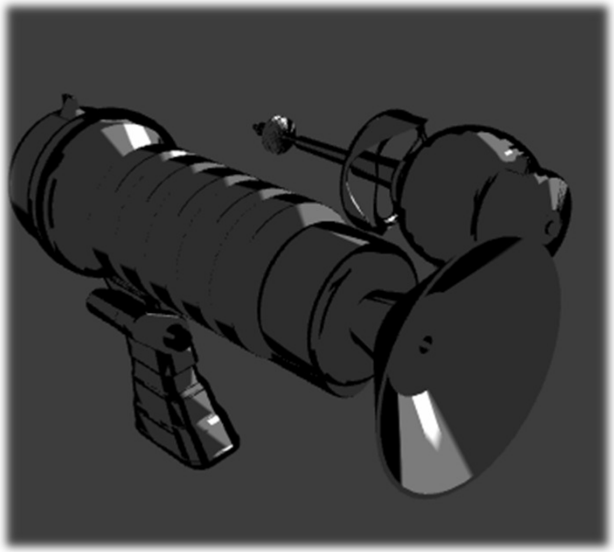
- A.k.a. Cel Shading ("Cel" is short for "celluloid" sheets, on which animation was hand-drawn)
- Gives any 3D model a cartoon-style look
- Emphasizes silhouettes
- Discrete steps for diffuse shading, highlights
- Non-photorealistic rendering method (NPR)
- Programmable shaders allow real-time performance



Source: Wikipedia



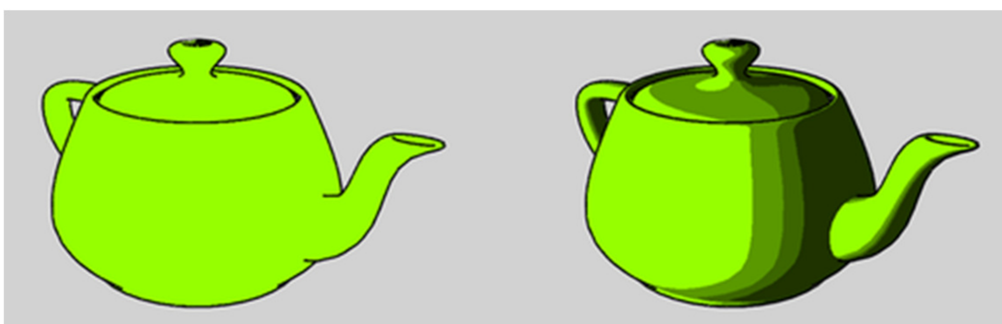
Off-line toon shader



GLSL toon shader

Approach

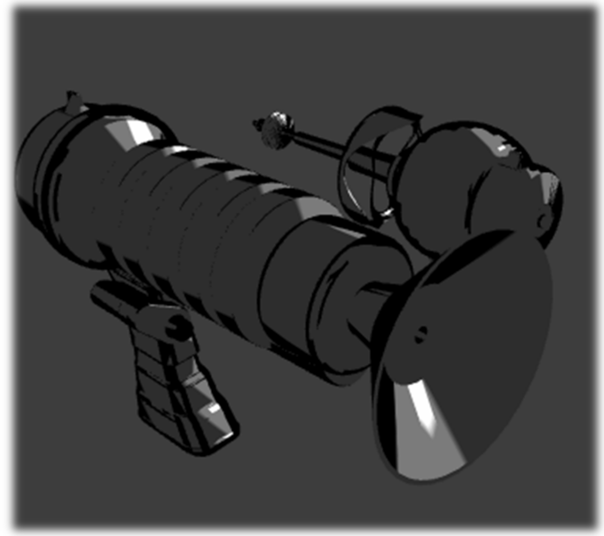
- Start with regular 3D model
- Apply two rendering tricks:
  - Silhouette edges
    - Emphasize pixels with normals perpendicular to viewing direction.
  - Discretized shading
    - Conventional (smooth) lighting values calculated for each pixel, then mapped to a small number of discrete shades.



Source: Wikipedia

Silhouette Edges

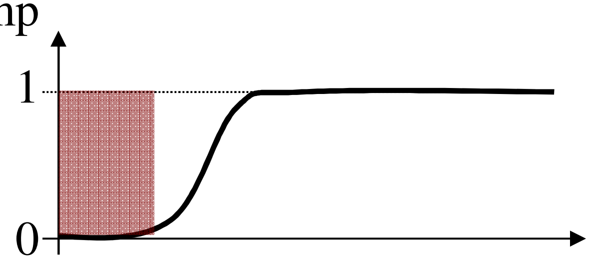
- Silhouette edge detection
  - Compute dot product of viewing direction $\mathbf{v}$ and normal $\mathbf{n}$



$$\text{edge} = \max(0, \mathbf{n} \cdot \mathbf{v})$$

  - Use 1D texture to define edge ramp

```
uniform sample1D edgeramp; e=texture1D(edgeramp,edge);
```



UCSD

Discretized Shading

- Compute diffuse and specular shading

$$\text{diffuse} = \mathbf{n} \cdot \mathbf{L} \qquad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s$$

- Use 1D textures diffuseramp, specularramp to map diffuse and specular shading to colors
- Final color:

```
uniform sampler1D diffuseramp;
uniform sampler1D specularramp;
c = e * (texture1D(diffuse,diffuseramp) +

texture1D(specular,specularramp));
```

Toon Shading Demo

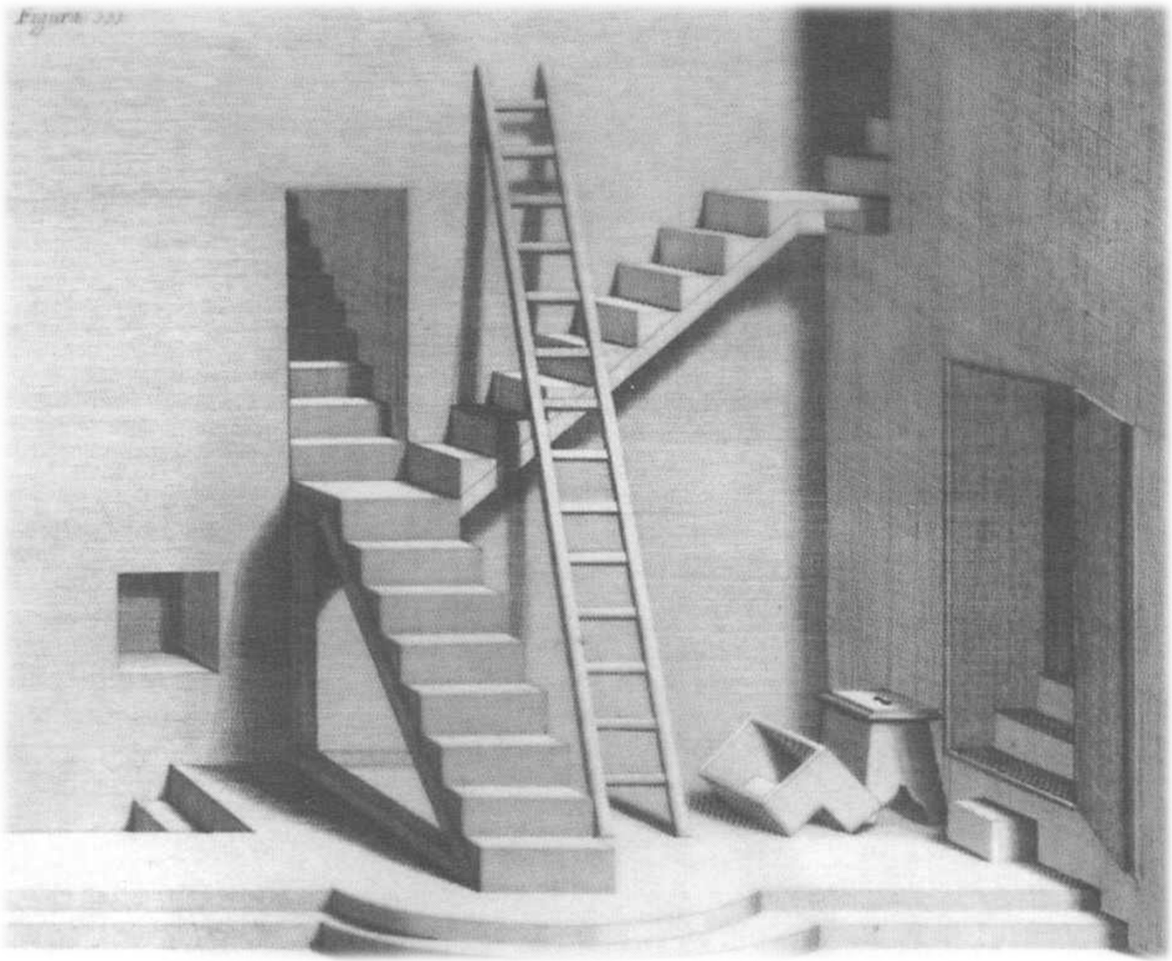
http://www.bonzaisoftware.com/npr.html

Lecture Overview

- Shadows
- Shadow Mapping

Why Are Shadows Important?

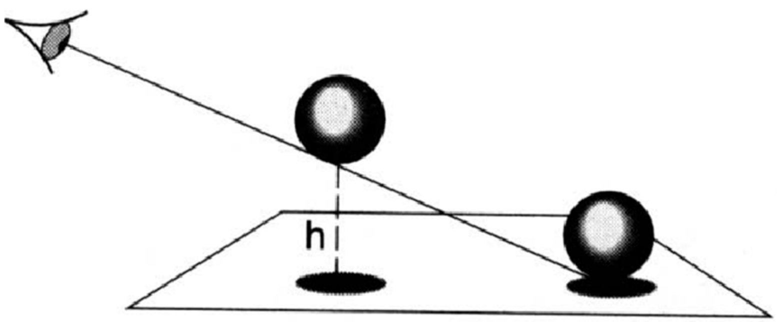
- Give additional cues on scene lighting

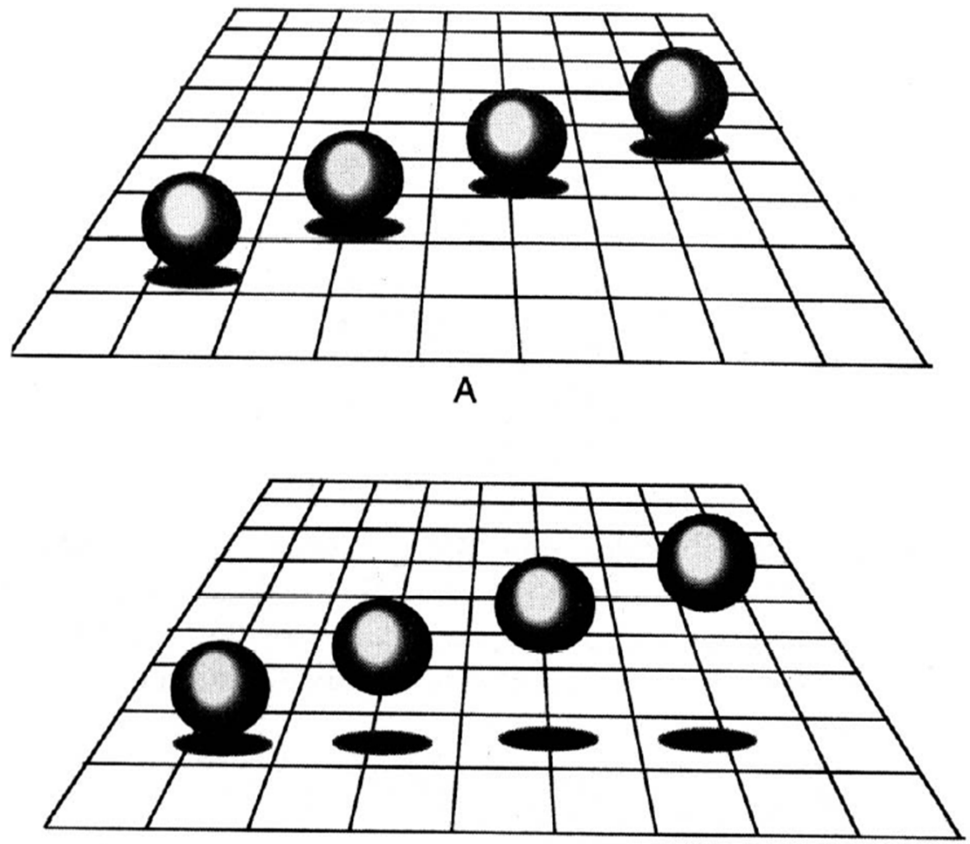


UCSD

Why Are Shadows Important?

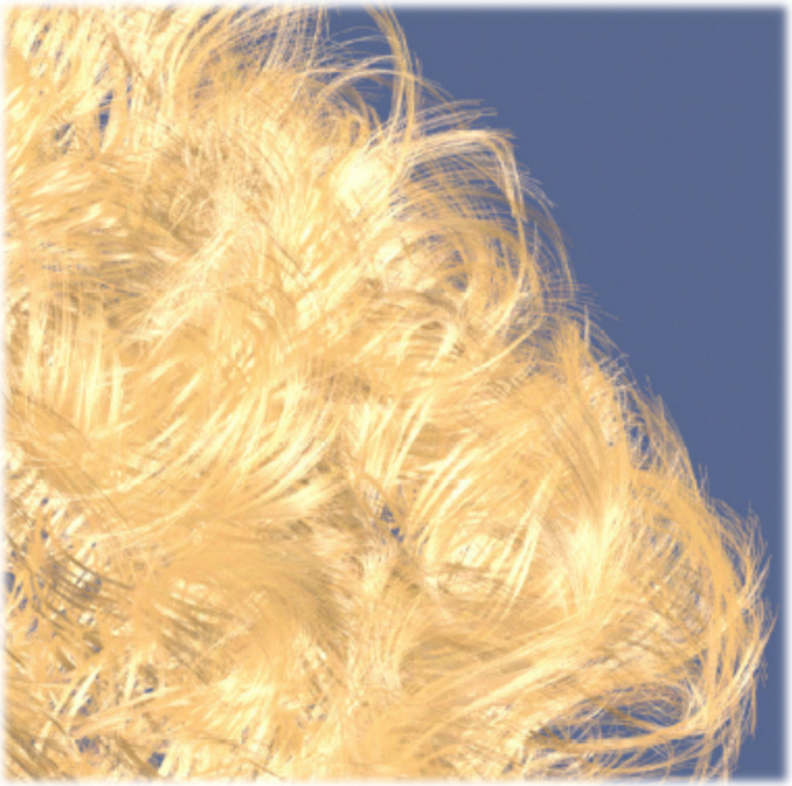
- Contact points
- Depth cues





Why Are Shadows Important?

- Realism

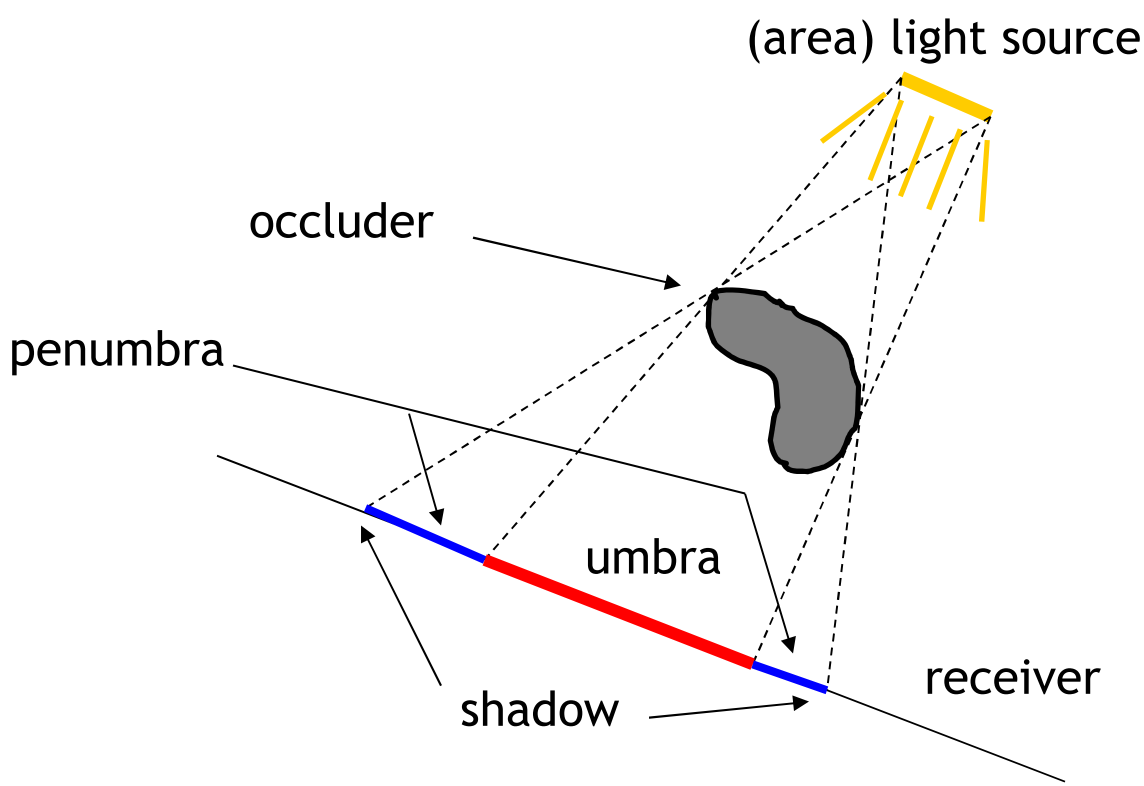


Without self-shadowing

With self-shadowing

Terminology

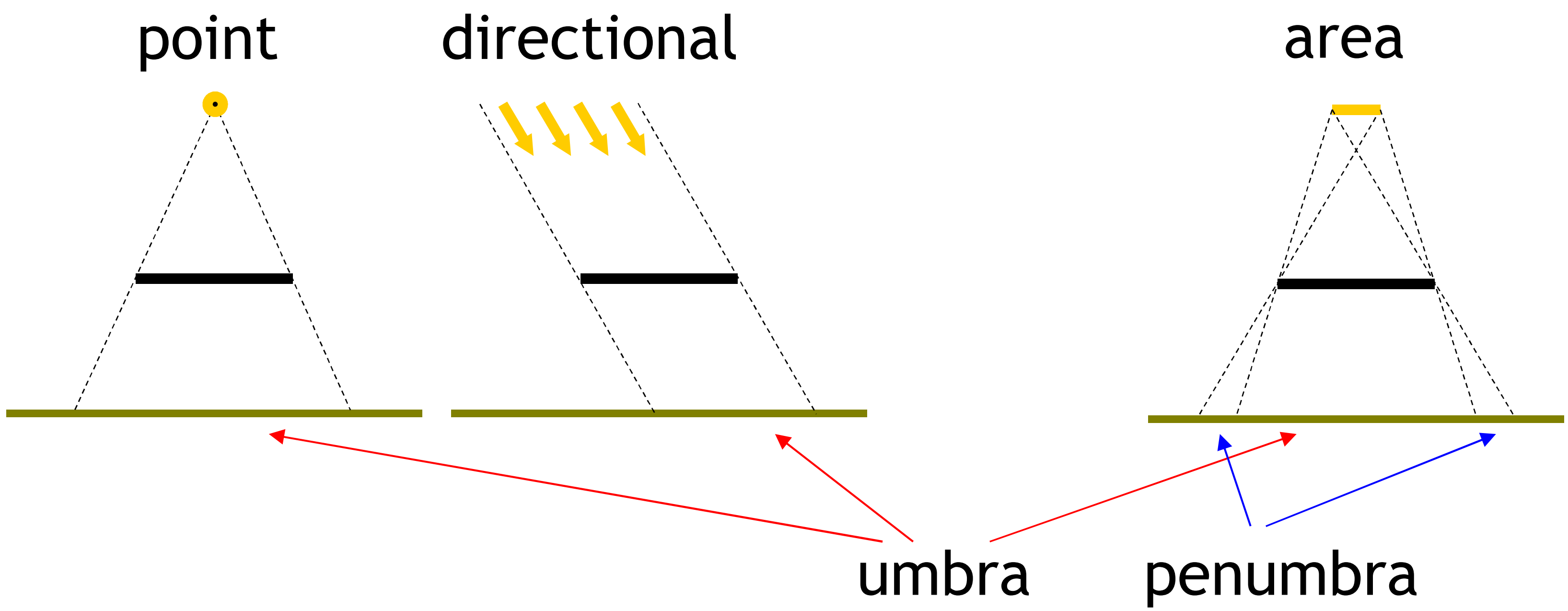
- Umbra: fully shadowed region
- Penumbra: partially shadowed region



UCSD

Hard and Soft Shadows

- Point and directional lights lead to hard shadows, no penumbra
- Area light sources lead to soft shadows, with penumbra



Hard and Soft Shadows

Hard shadow from point light source

Soft shadow from area light source

Shadows for Interactive Rendering

- In this course: hard shadows only
  - Soft shadows hard to compute in interactive graphics
- Two most popular techniques:
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Active research area

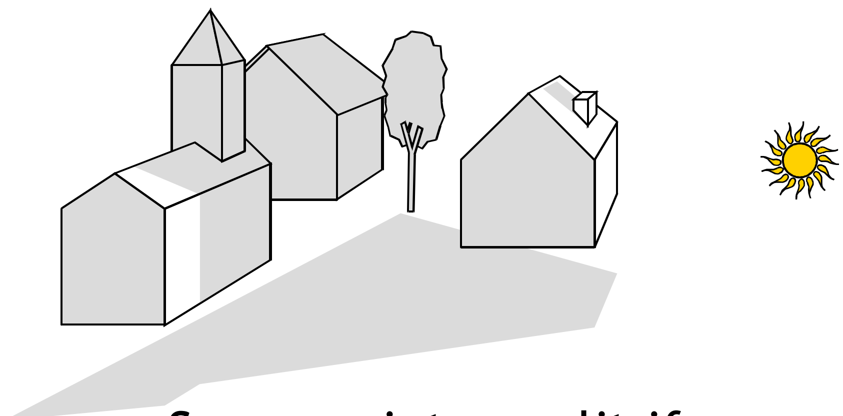
Lecture Overview

- Shadows
- Shadow Mapping

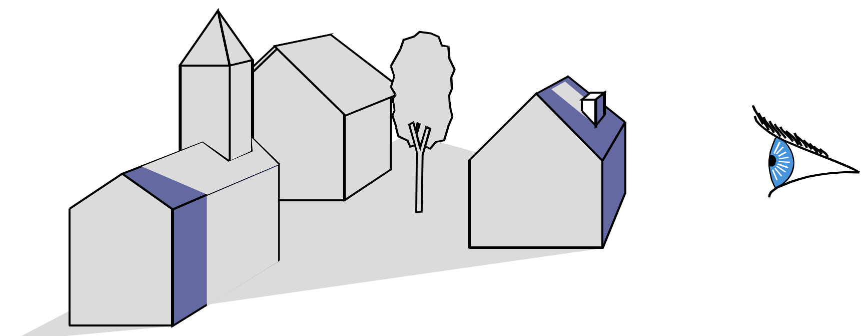
Shadow Mapping

Main Idea

- A scene point is lit by the light source if visible from the light source
- Determine visibility from light source by placing a camera at the light source position and rendering the scene from there



Scene points are lit if visible from light source



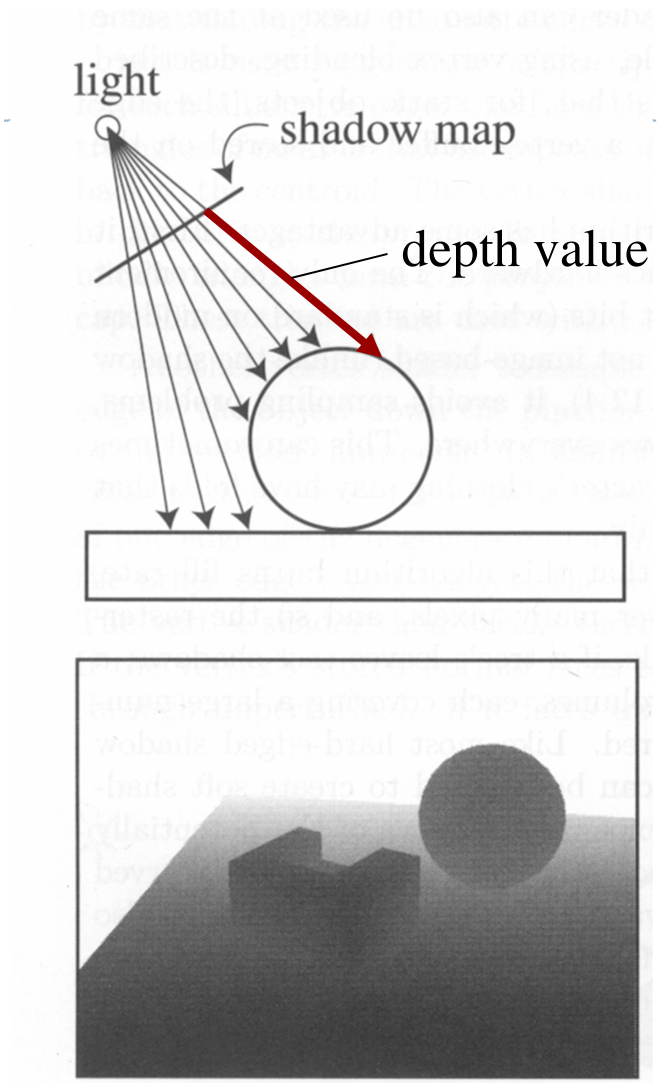
Determine visibility from light source by placing camera at light source position

UCSD

Two Pass Algorithm

First Pass

- Render scene by placing camera at light source position
- Store depth image (*shadow map*)



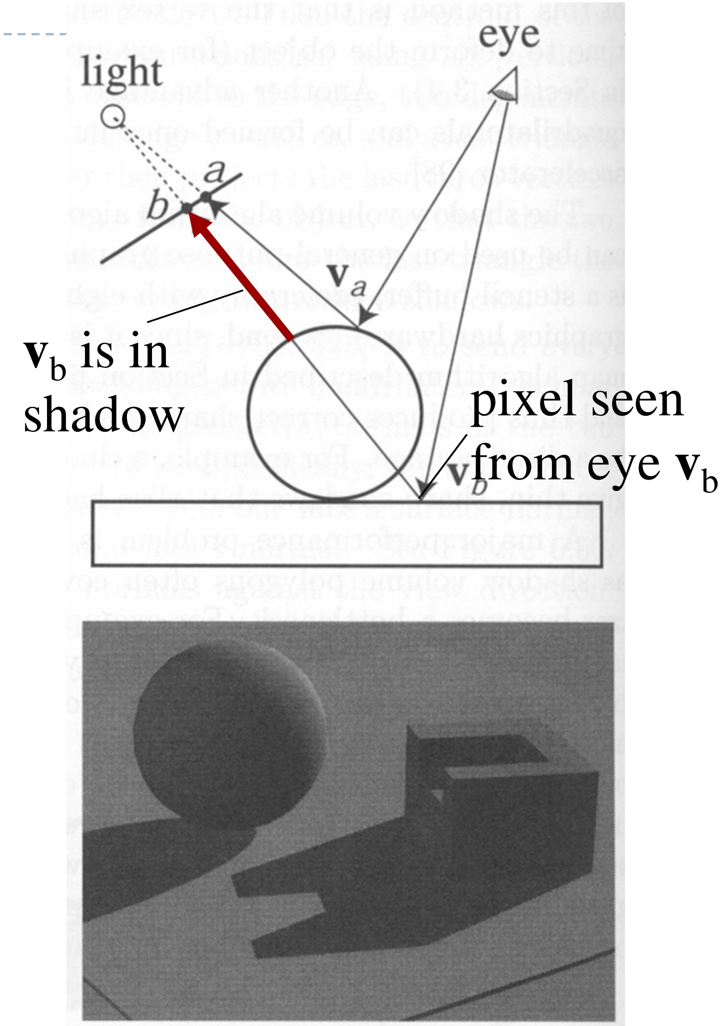
Depth image as seen from light source

UCSD

Two Pass Algorithm

Second Pass

- Render scene from camera position
- At each pixel, compare distance to light source with value in shadow map
  - If distance is larger, pixel is in shadow
  - If distance is smaller or equal, pixel is lit



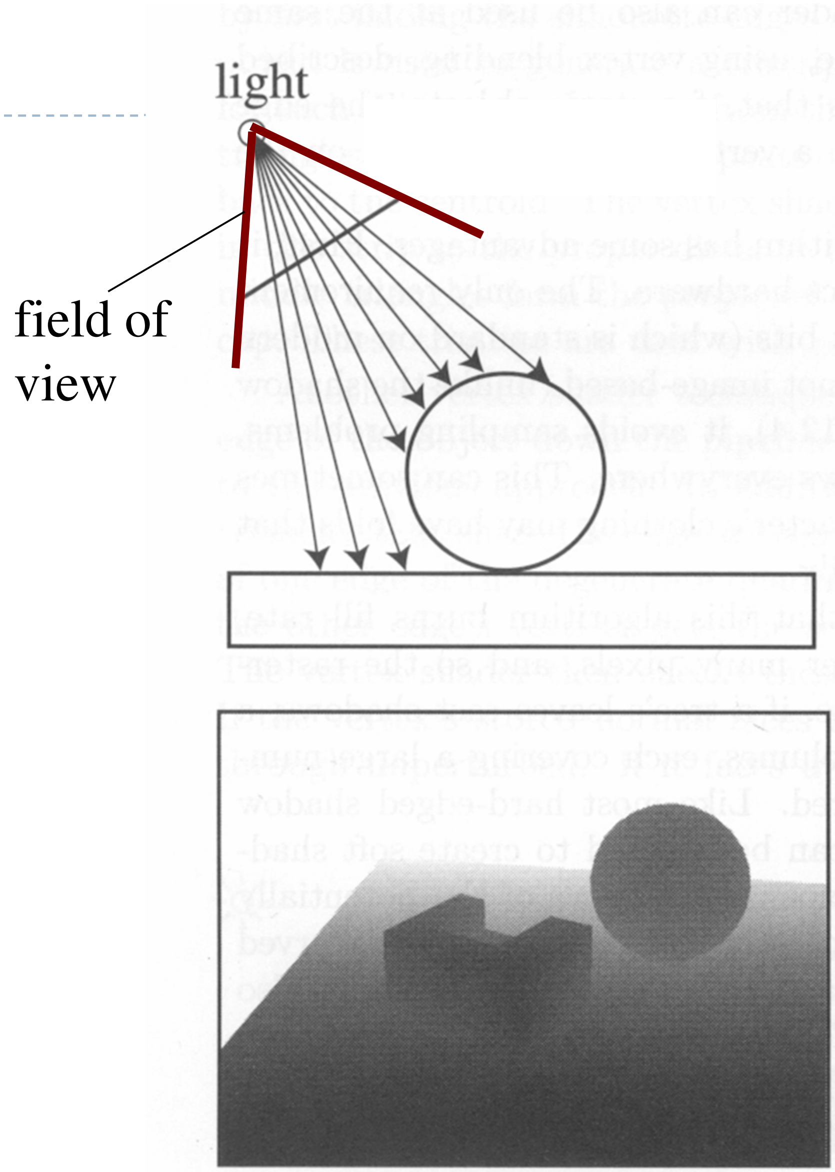
Final image with shadows

Issues With Shadow Maps

- Limited field of view of shadow map
- Z-fighting
- Sampling problems

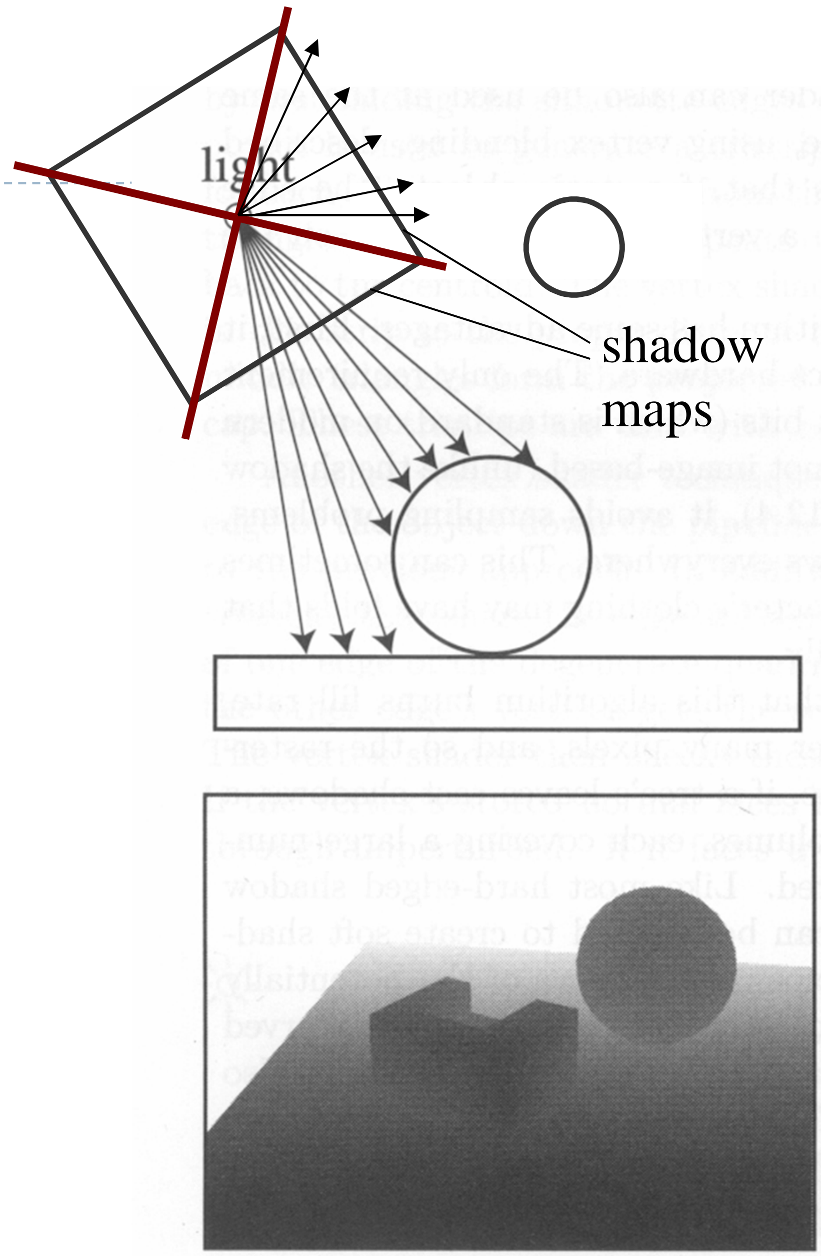
Limited Field of View

- What if a scene point is outside the field of view of the shadow map?



Limited Field of View

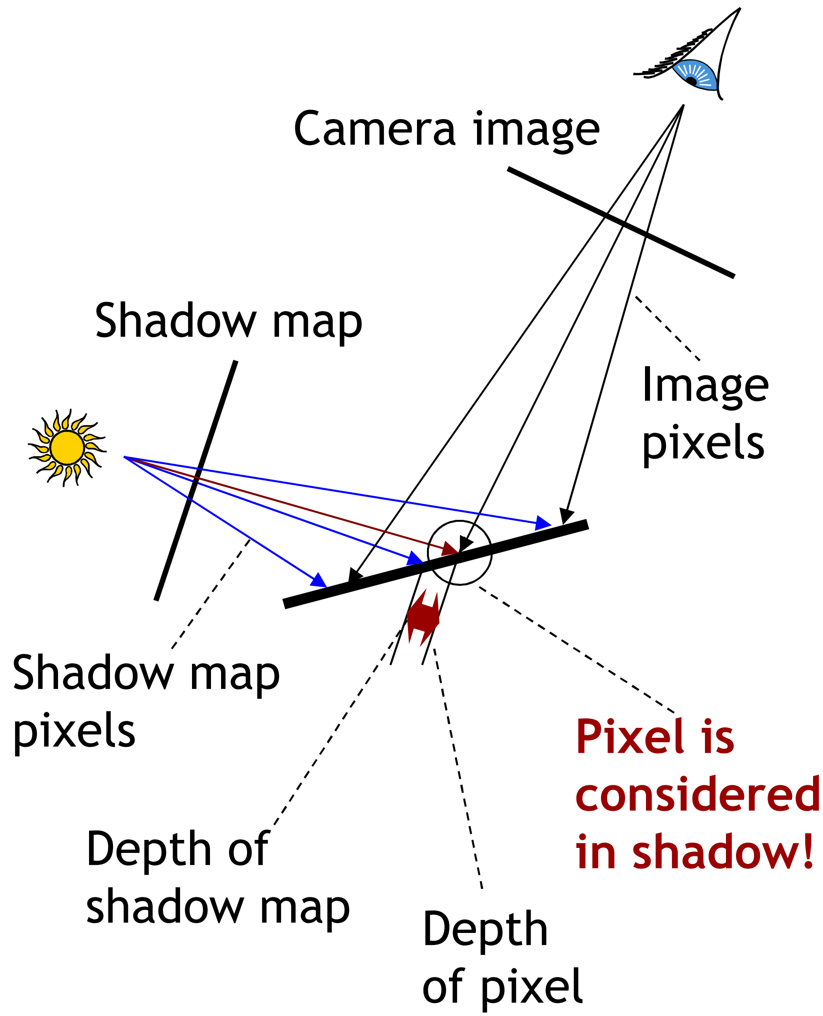
- What if a scene point is outside the field of view of the shadow map?
  → Use six shadow maps, arranged in a cube
- Requires a rendering pass for each shadow map



UCSD

Z-Fighting

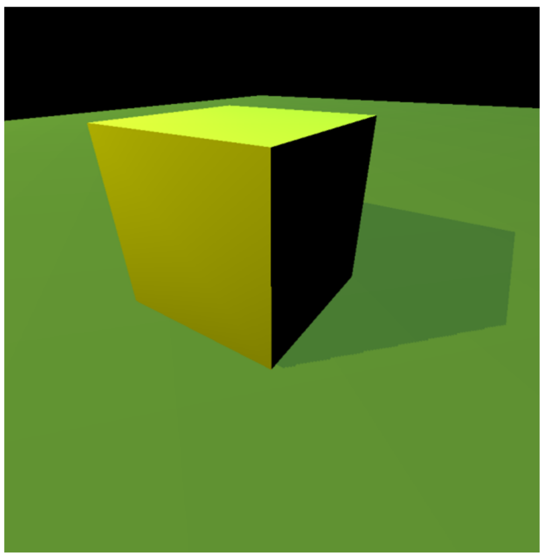
- Depth values for points visible from light source are equal in both rendering passes
- Because of limited resolution, depth of pixel visible from light could be larger than shadow map value
- Need to add bias in first pass to make sure pixels are lit



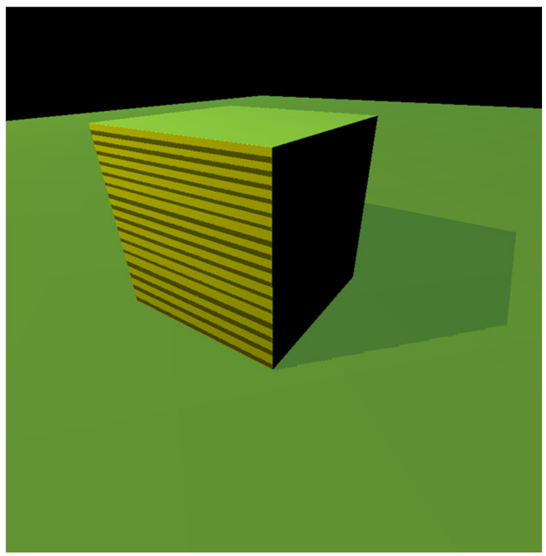
UCSD

Solution: Bias

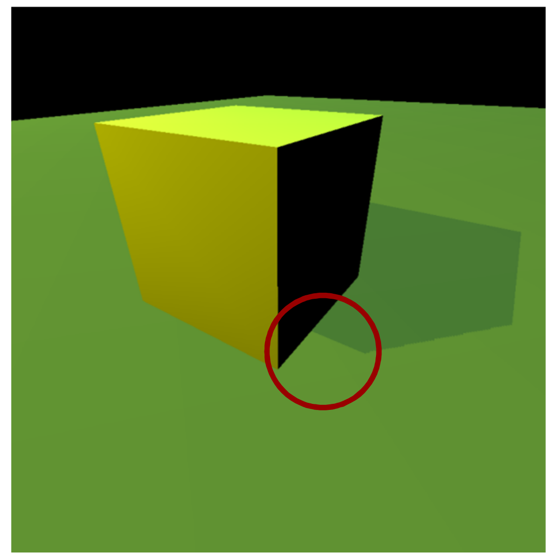
- Add bias when rendering shadow map
  - Move geometry away from light by small amount
- Finding correct amount of bias is tricky



Correct bias



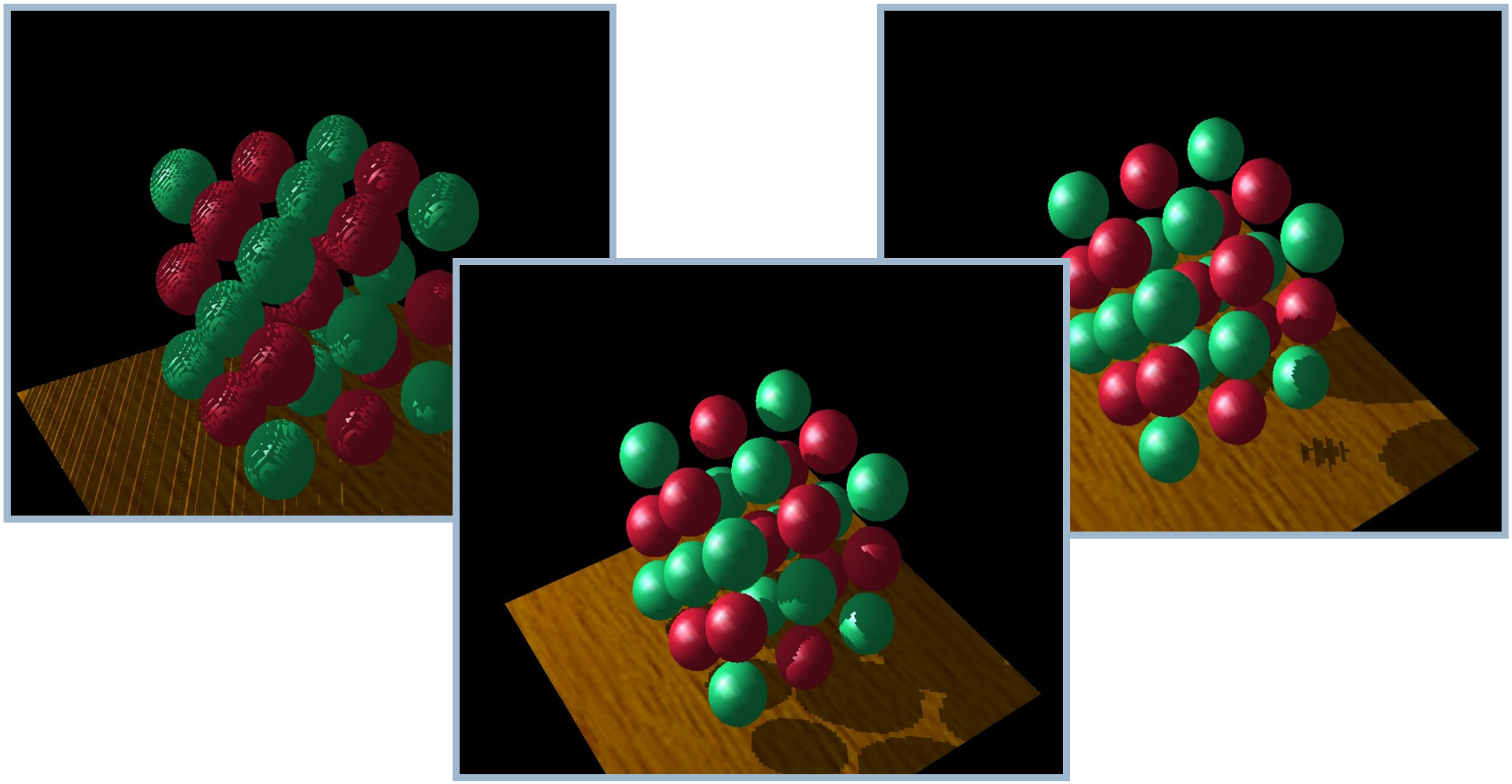
Not enough bias



Too much bias

Bias Adjustment

Not enough | Too much

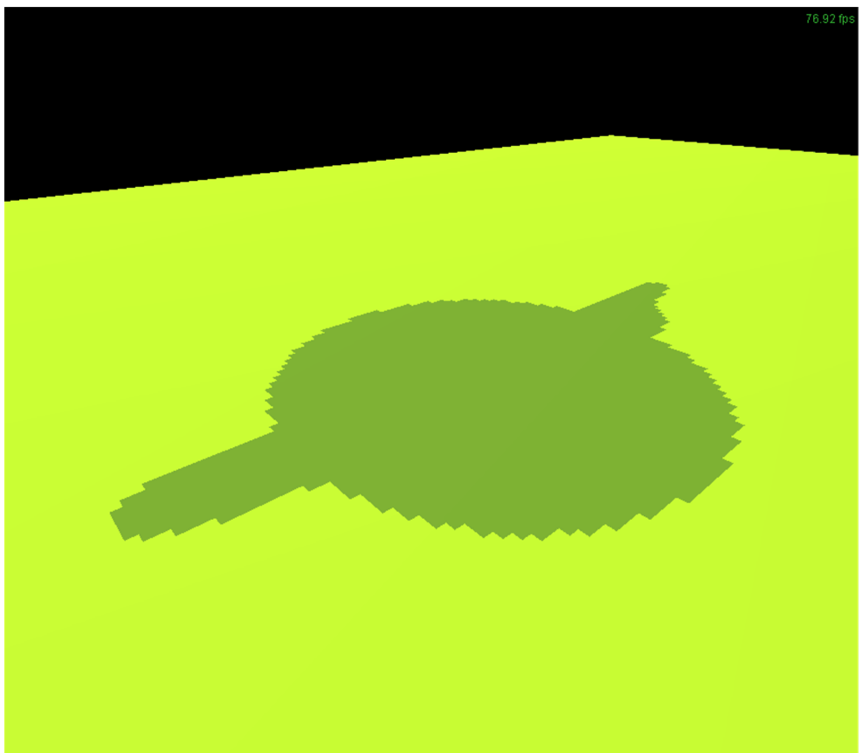


Just right

UCSD

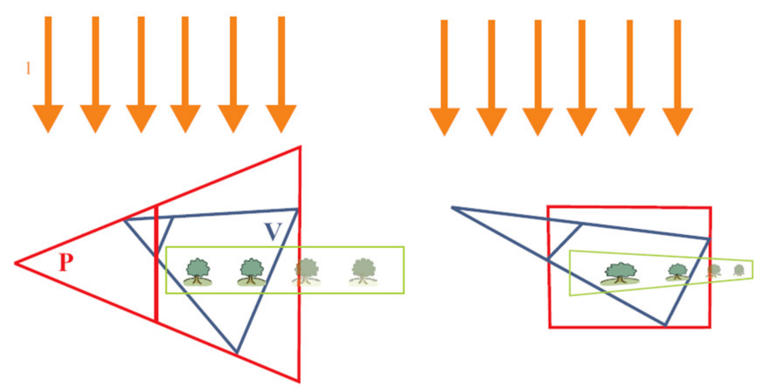
Sampling Problems

- Shadow map pixel may project to many image pixels
  → Stair-stepping artifacts



Solutions

- Increase resolution of shadow map
  - Not always sufficient
- Split shadow map into several tiles
- Tweak projection for shadow map rendering
  - Light space perspective shadow maps (LiSPSM) http://www.cg.tuwien.ac.at/research/vr/lispsm/



- Combination of splitting and LiSPSM
  - Basis for most serious implementations

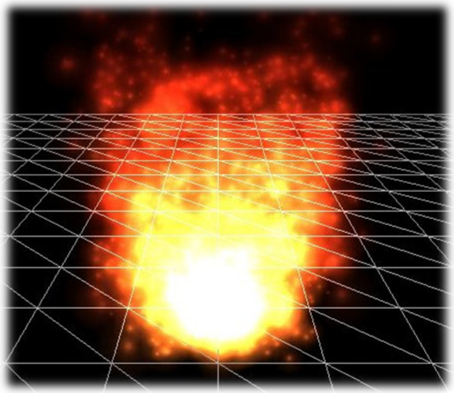
UCSD

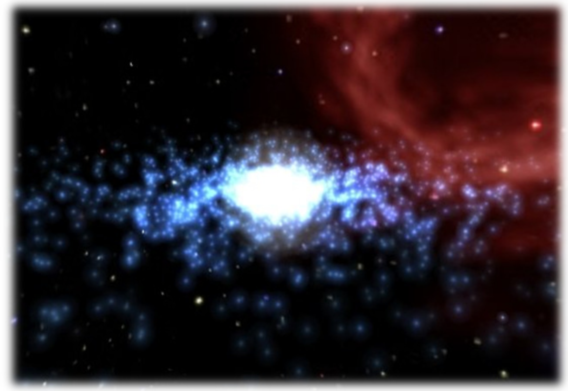
Lecture Overview

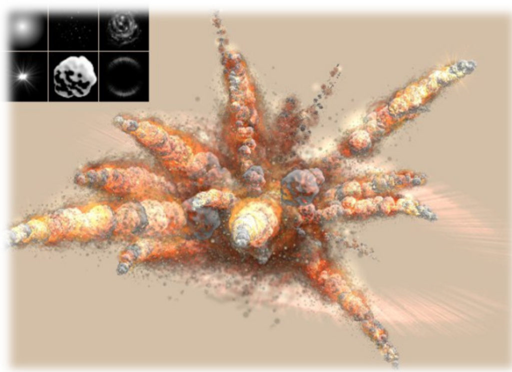
- Particle Systems
- Collision Detection
- Deferred Rendering

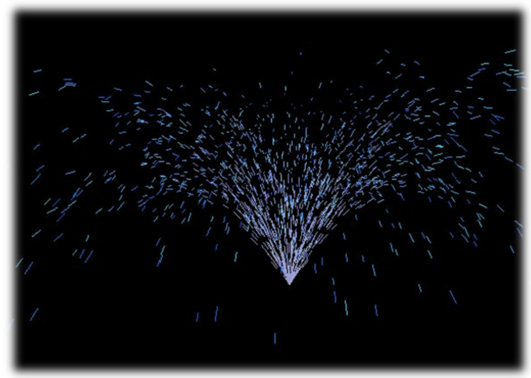
Particle Systems

- Used for:
  - Fire/sparks
  - Rain/snow
  - Water spray
  - Explosions
  - Galaxies







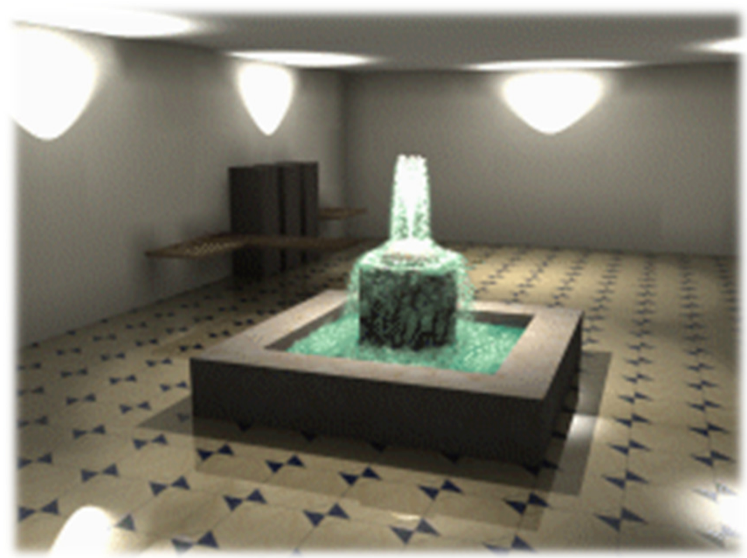


Internal Representation

- Particle system is collection of a number of individual elements (particles)
  - Controls a set of particles which act autonomously but share some common attributes
- Particle Emitter: Source of all new particles
  - 3D point
  - Polygon mesh: particles' initial velocity vector is normal to surface
- Particle attributes:
  - position (3D)
  - velocity (vector: speed and direction)
  - color + opacity
  - lifetime
  - size
  - shape
  - weight

Dynamic Updates

- Particles change position and/or attributes with time
- Initial particle attributes often created with random numbers
- Frame update:
  - Parameters: simulation of particles, can include collisions with geometry
    - Forces (gravity, wind, etc) accelerate a particle
    - Acceleration changes velocity
    - Velocity changes position
  - Rendering: display as
    - OpenGL points
    - (Textured) billboarded quads
    - Point sprites



Source: http://www.particlesystems.org/

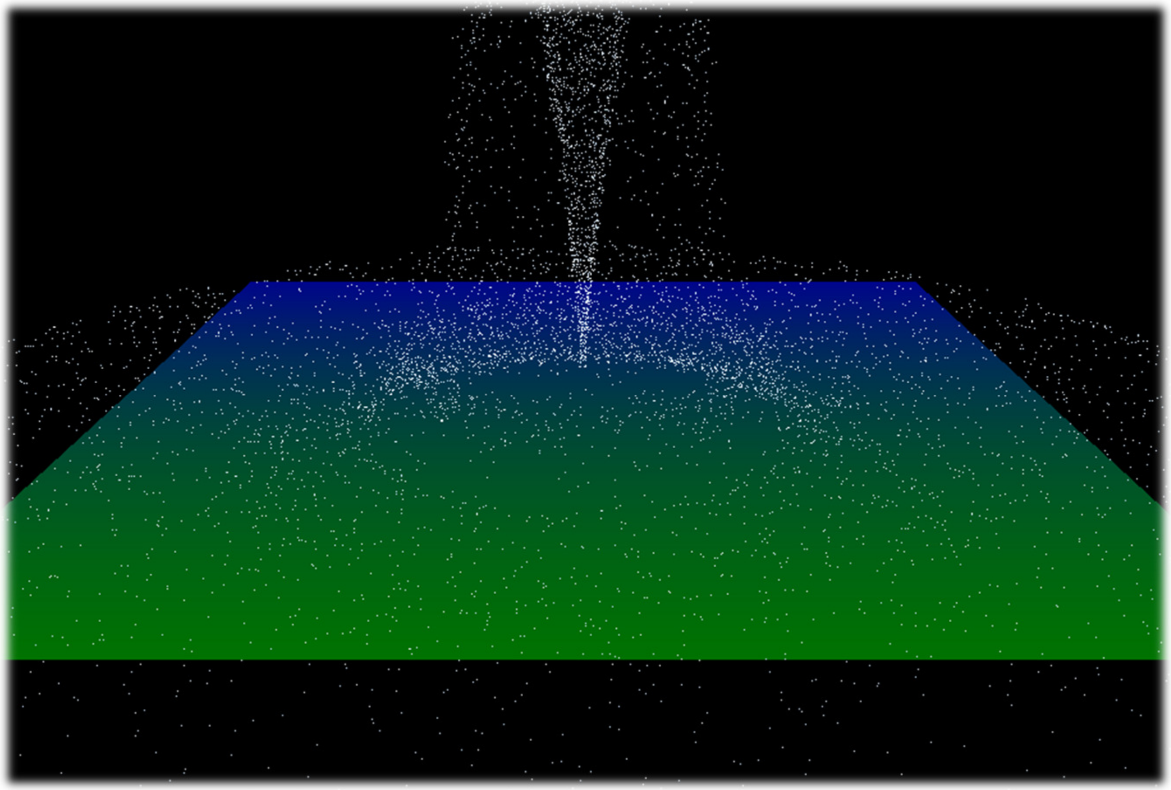
Point Sprite

- **Screen-aligned element of variable size**
- **Defined by single point**
- **Sample code:**

```
glTexEnvf(GL_POINT_SPRITE, GL_COORD_REPLACE, GL_TRUE);
glEnable(GL_POINT_SPRITE);
glBegin(GL_POINTS);
  glVertex3f(position.x, position.y, position.z);
glEnd();
glDisable(GL_POINT_SPRITE);
```

Demo

- Demo software by Prof. David McAllister:
  - http://www.calit2.net/~jschulze/tmp/Particle221Demos.zip



References

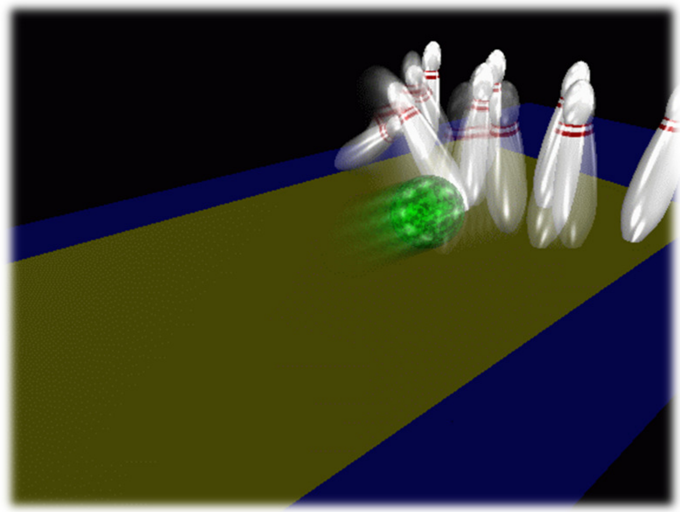
- Tutorial with source code by Bartlomiej Filipek, 2014:
  - http://www.codeproject.com/Articles/795065/Flexible-particle-system-OpenGL-Renderer
- Articles with source code:
  - Jeff Lander: "The Ocean Spray in Your Face", Game Developer, July 1998
    - http://www.darwin3d.com/gamedev/articles/col0798.pdf
  - John Van Der Burg: "Building an Advanced Particle System", Gamasutra, June 2000
    - http://www.gamasutra.com/view/feature/3157/building_an_advanced_particle_.php
- Founding scientific paper:
  - Reeves: "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions on Graphics (TOG) Volume 2 Issue 2, April 1983
    - http://zach.in.tu-clausthal.de/teaching/vr_literatur/Reeves%20-%20Particle%20Systems.pdf

Lecture Overview

- Particle Systems
- Collision Detection
- Deferred Rendering

Collision Detection

- Goals:
  - Physically correct simulation of collision of objects
    - Not covered here
  - Determine if two objects intersect
- Slow calculation because of exponential growth $O(n^2)$:
  - # collision tests = n*(n-1)/2



UCSD

Intersection Testing

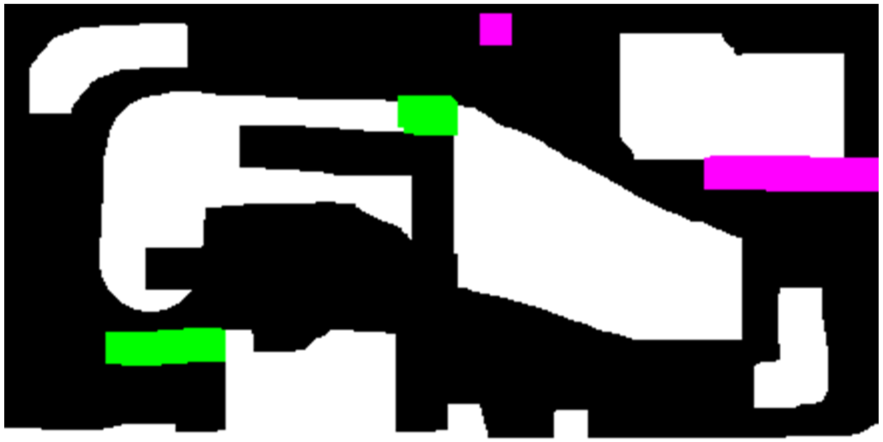
- Purpose:
  - Keep moving objects on the ground
  - Keep moving objects from going through walls, each other, etc.
- Goal:
  - Believable system, does not have to be physically correct
- Priority:
  - Computationally inexpensive
- Typical approach:
  - Spatial partitioning
  - Object simplified for collision detection by one or a few
    - Points
    - Spheres
    - Axis aligned bounding box (AABB)
  - Pairwise checks between points/spheres/AABBs and static geometry

Sweep and Prune Algorithm

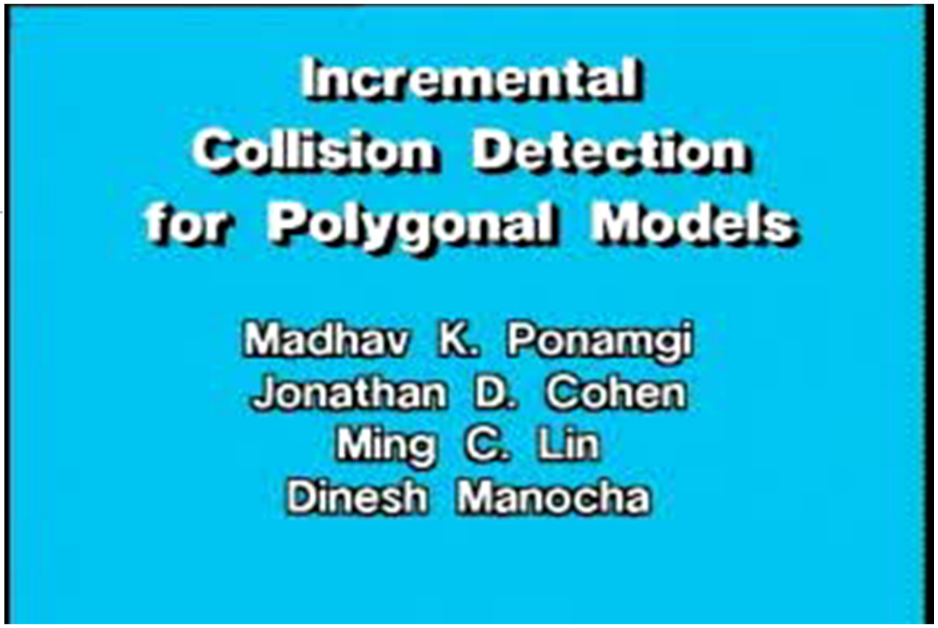
- Sorts bounding boxes
- Not intuitively obvious how to sort bounding boxes in 3-space
- Dimension reduction approach:
  - Project each 3-dimensional bounding box onto the x,y and z axes
  - Find overlaps in 1D: a pair of bounding boxes can overlap if and only if their intervals overlap in all three dimensions
    - Construct 3 lists, one for each dimension
    - Each list contains start/end point of intervals corresponding to that dimension
    - By sorting these lists, we can determine which intervals overlap
    - Reduce sorting time by keeping sorted lists from previous frame, changing only the interval endpoints
- Alternative: project bounding boxes onto coordinate axis planes and look for overlaps in 2D

Collision Map (CM)

- 2D map with information about where objects can go and what happens when they go there
- Colors indicate different types of locations
- Map can be computed from 3D model, or hand drawn with paint program
- Granularity: defines how much area (in object space) one CM pixel represents



References



- I-Collide:
  - Interactive and exact collision detection library for large environments composed of convex polyhedra
    - http://gamma.cs.unc.edu/I-COLLIDE/
- OZ Collide:
  - Fast, complete and free collision detection library in C++
  - Based on AABB tree
    - http://www.tsarevitch.org/ozcollide/