Shadow maps/Skuggmappning

Mycket populär skuggningsmetod!

- Två renderingar av scenen
- Beräkning för beslut i fragmentshader
- Mycket beräkningar (filter) för god kvalitet

Fördel: Behöver ingen kunskap om scenens innehåll, klarar alla sorters former.

Occluder och receiver hittas automatiskt! “Self-shadowing” inget problem.

Shadow maps

Tvåpassalgoritm:

1: Rendera från ljuskällan. Endast Z-buffern är av intresse! Z-buffern är vår “shadow map”, en 2D-funktion som anger avståndet till ljuskällan.

2: Z-buffern används i andra passet:

Z-buffern används som projicerad textur, men inte för att rita med!

Shadow maps

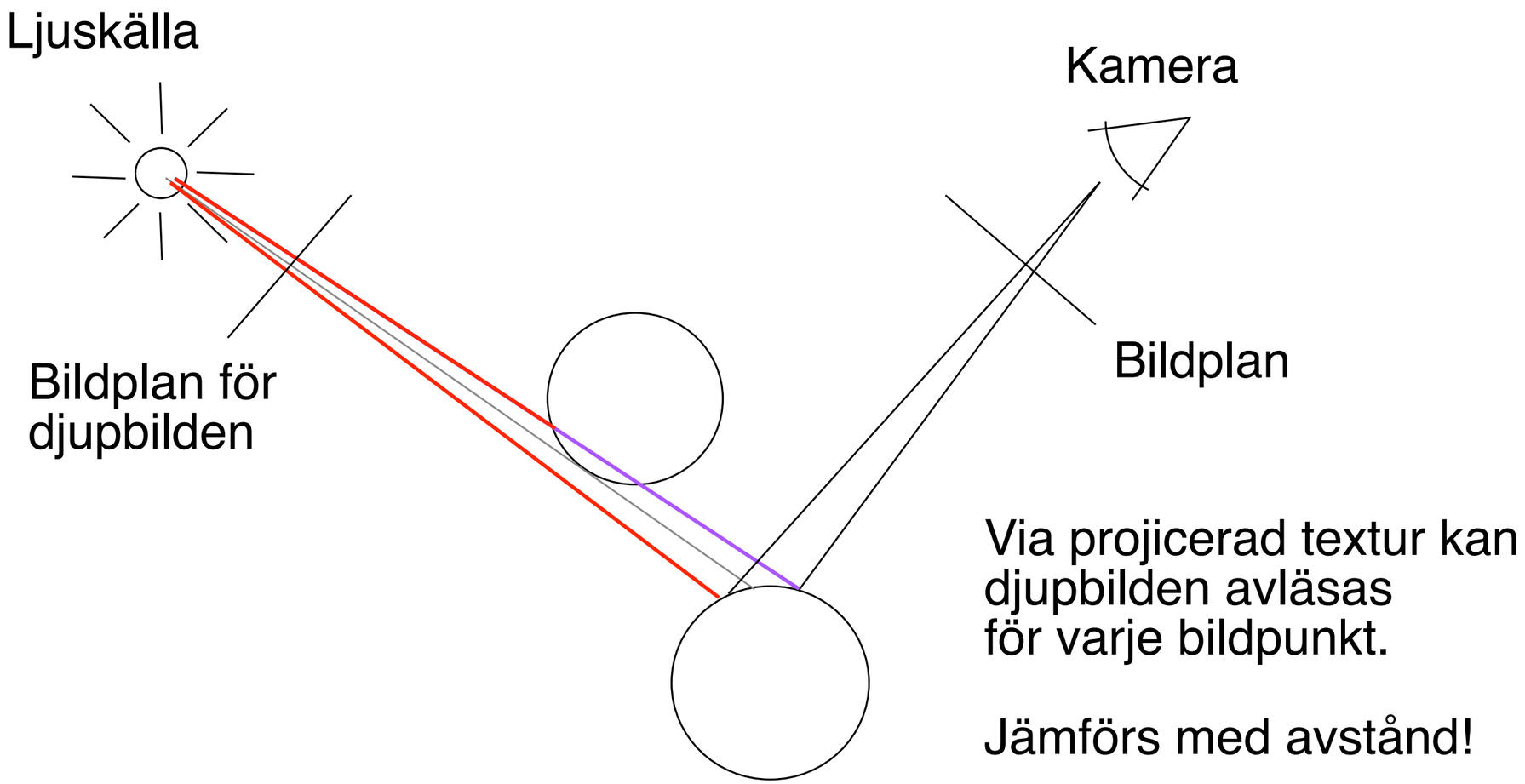
Z-buffern är en “djuptextur”. När den projiceras över scenen så nås varje fragmentberäkning av ett Z-värde.

A: Om fragmentet (pixeln) är belyst så är Z-bufferns värde lika med avståndet till ljuskällan

B: Om den är i skugga så är Z-värdet mindre än avståndet till ljuskällan.

Om vi kan testa denna likhet/olikhet så vet vi om pixeln är i skugga! Detta görs med fragmentshadern.

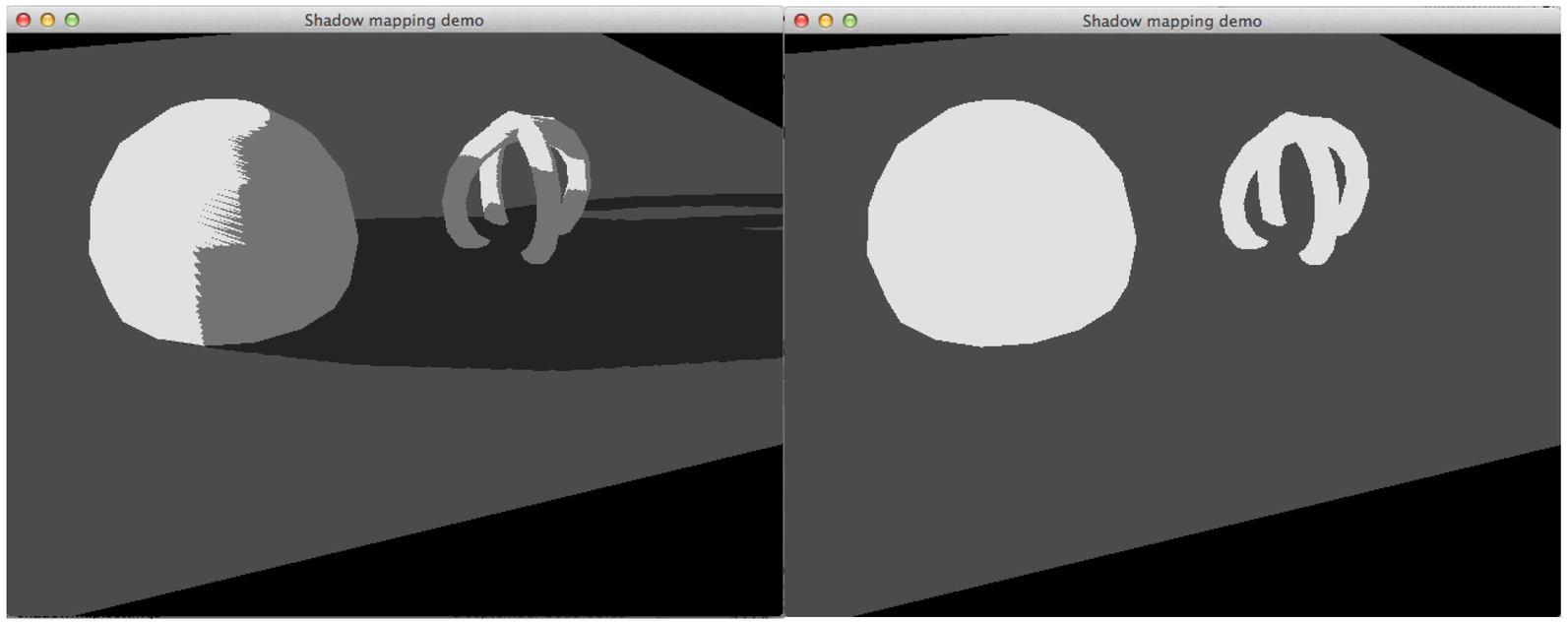
Shadow maps - princip



Via projicerad textur kan djupbilden avläsas för varje bildpunkt.

Jämförs med avstånd!

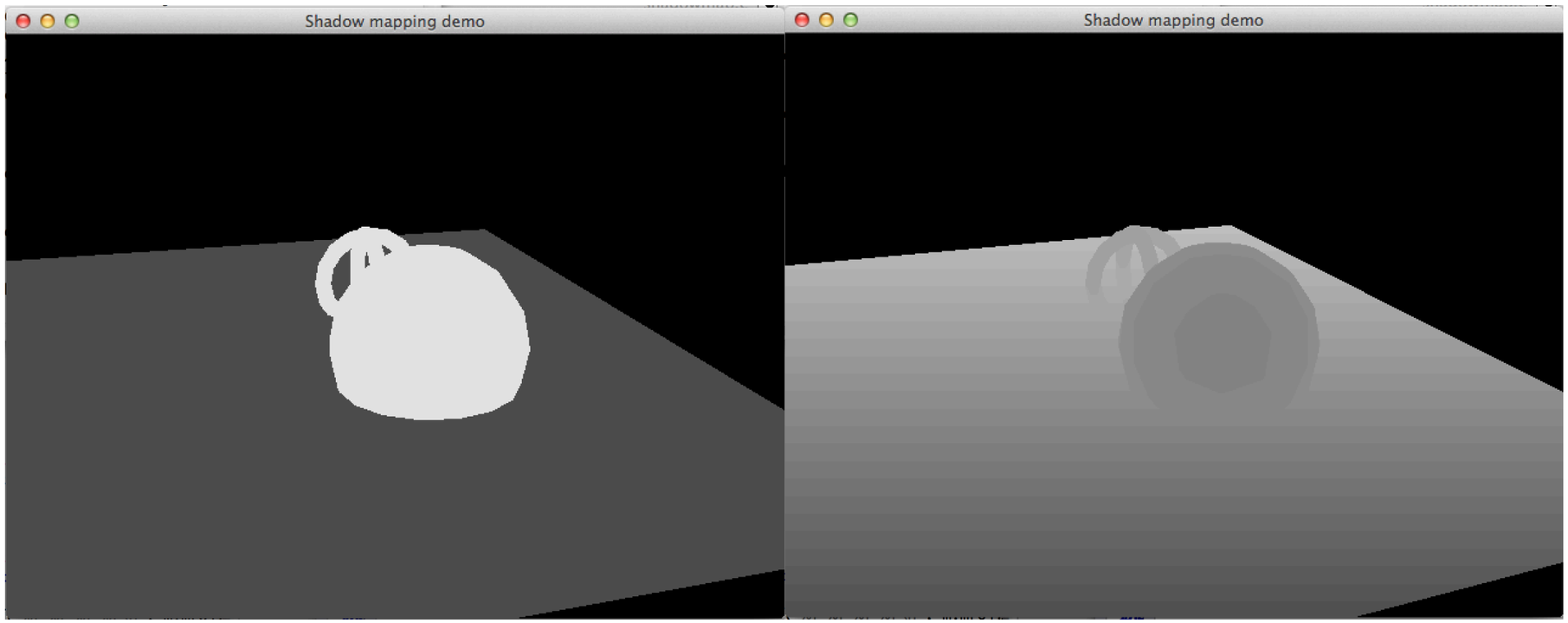
Shadow maps - exempel



Med skuggor Utan skuggor

Shadow maps - exempel

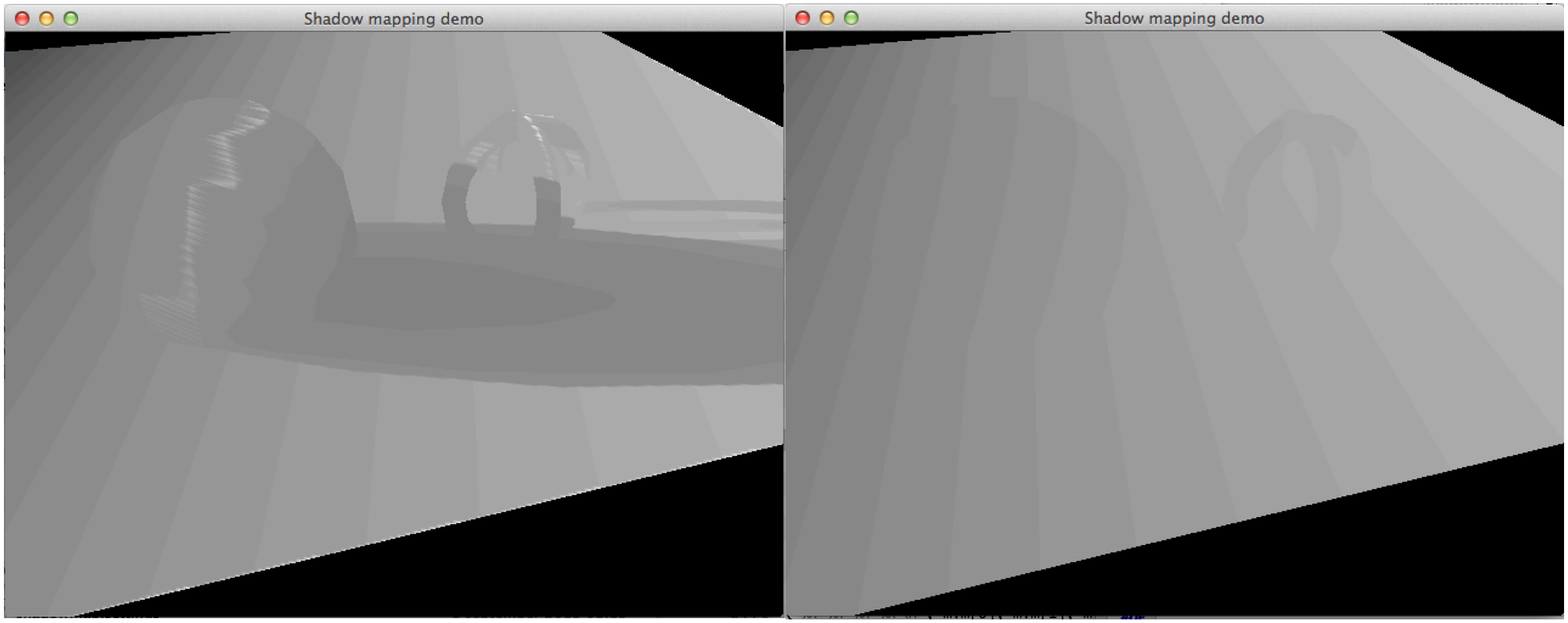
Rendering av shadow map



Sedd från ljuskällan Z-buffer från ljuskällan

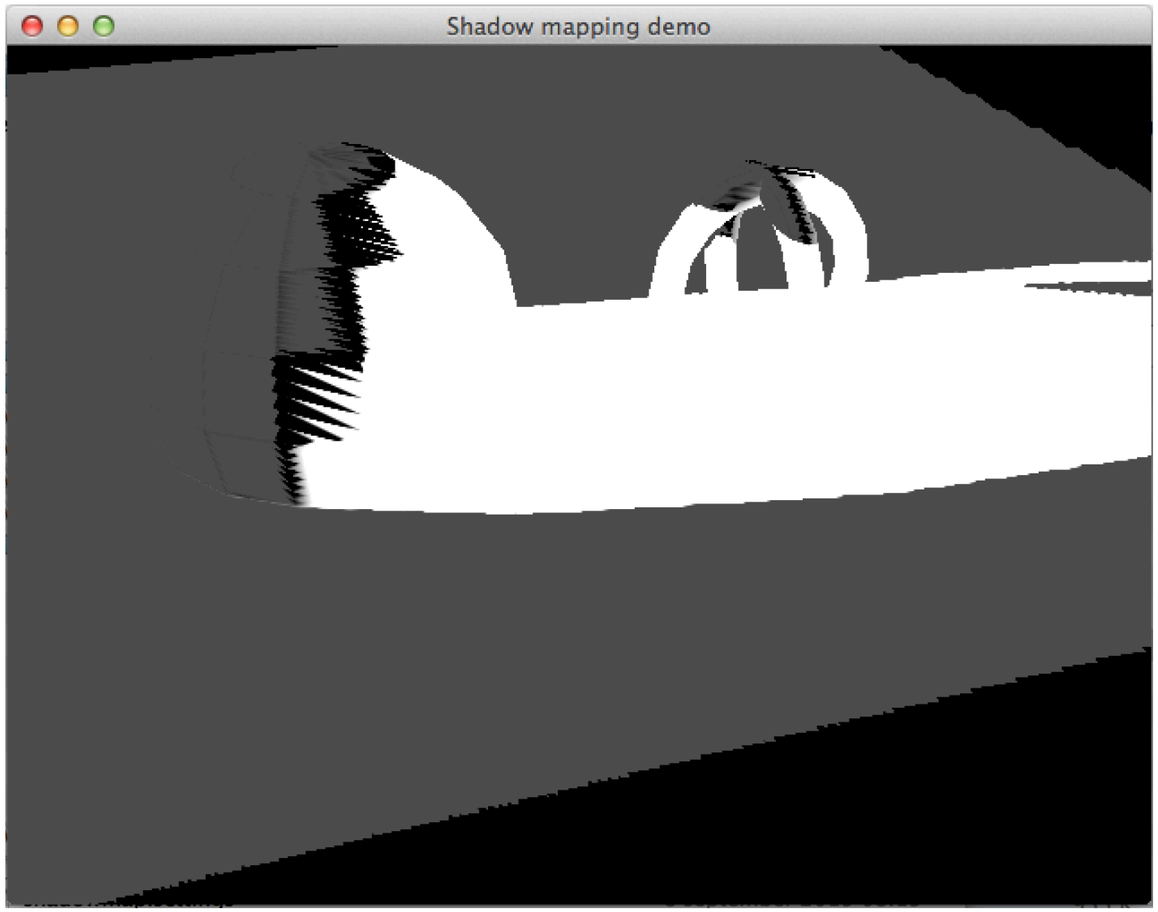
Shadow maps - exempel

Shadow map som projicerad textur



När dessa är lika: ej i skugga!

Områden med avvikande värden:



Shadow maps - problem

Problem 1: “Lika” är ett dåligt villkor

Måste ha en viss maginal för att undvika artefakter.
För lite marginal: Ytor blir delvis skuggade (liknar Z-fightning)
För mycket marginal: Skuggor krymper

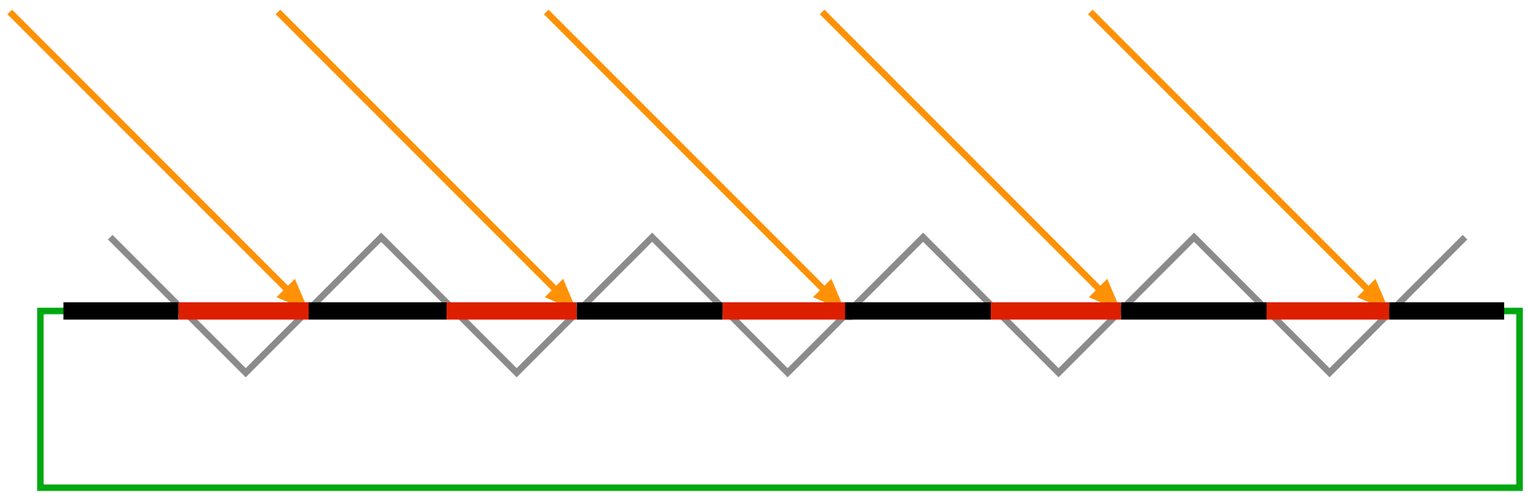
Problem 2: Shadow buffer kräver hög upplösning för att inte ge kantartefakter

Shadow acne

eller *erroneous self-shadowing*

"Trappstegseffekt". Skuggmap *hitom* ytan = skugga.

Störst problem längs sidorna, stark lutning sett från ljuskällan.



Svart = skugga. Rött = ej skugga.

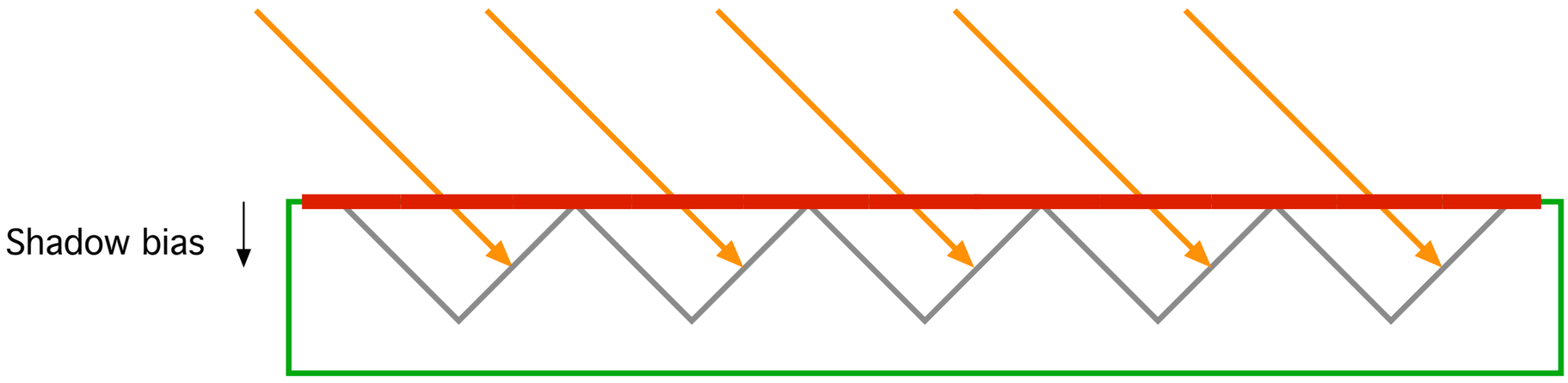
Bild baserad på bild från LearnOpenGL

Shadow bias & Peter panning

"Åtgärd: Offset, *shadow bias*.

Skuggmappens yta räknas som *längre bort*.

Problem 2: För stor bias kan synas som förskjutna skuggor.



Svart = skugga. Rött = ej skugga.

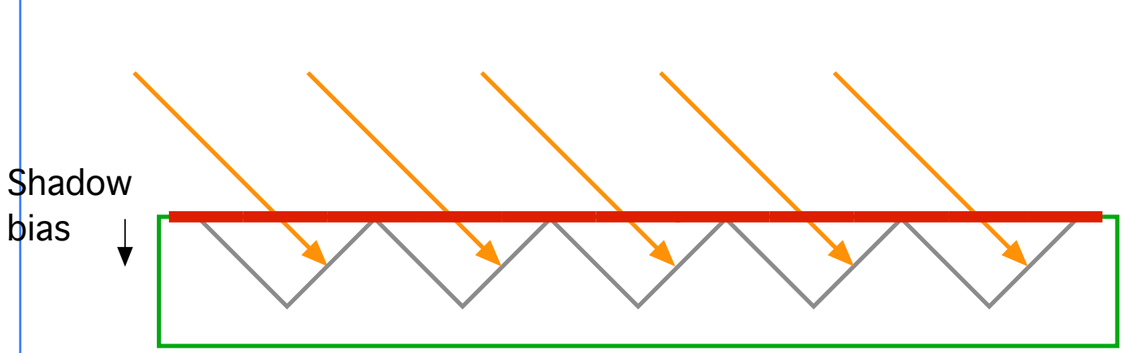
Bild baserad på bild från LearnOpenGL

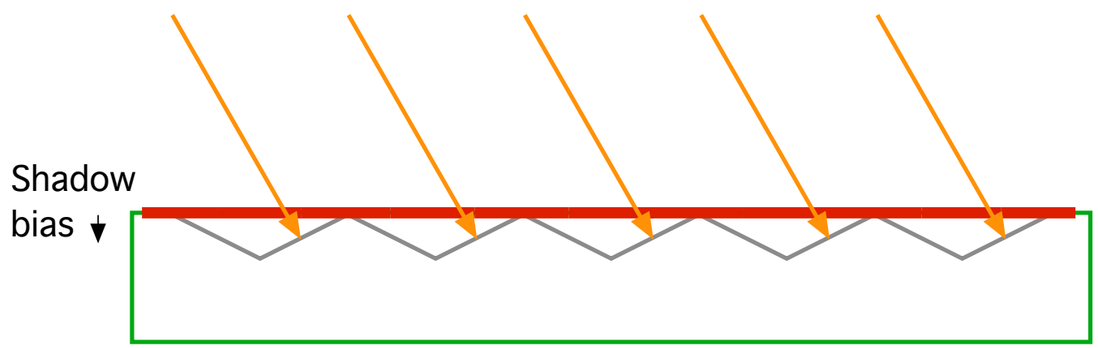
Lutningsberoende shadow bias

Störst problem i brant lutning.

Ha olika mycket shadow bias beroende på lutning!

Minskar Peter Panning



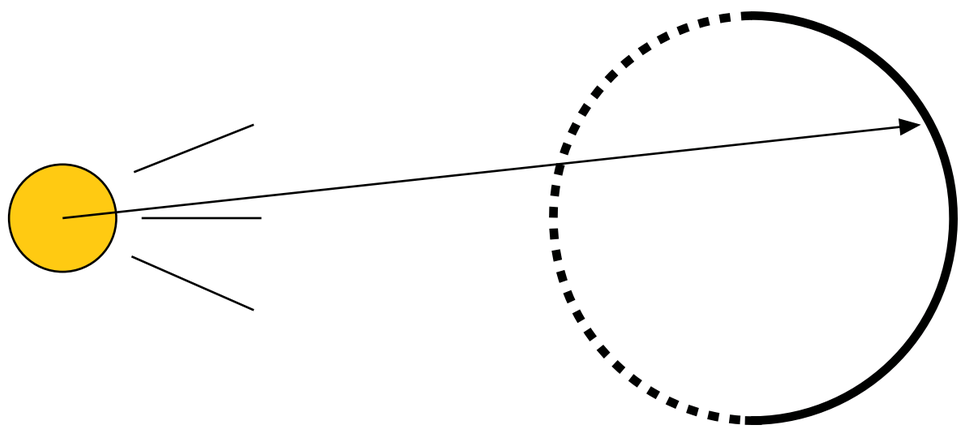


Front-face culling

Trick för att undvika shadow acne.

Mät djup till *baksidan* i stället för framsidan.

Flyttar problemet till baksidan - men eliminerar det inte.



Shadow buffer-exempel

Baserat på demo av Fabien Saglard, förenklat, generaliserat.

Shaderbaserat. (Fixed pipeline-lösningar fanns också men var mindre flexibelt.)

Shadow buffer-exempel

Viktiga detaljer

- Placera kameran i ljuskällan
- Z-buffer till textur
- Projicera textur
- Utför avståndsjämförelse

Z-buffer till textur

Kan göras med kopiering mellan buffrar.

```
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 0,0, width, height,0);
```

Nya, snabbare metoden: FBO!

Skapa FBO med enbart Z-buffer!

Lite annorlunda FBO; stäng av de vanliga delarna:

```
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
```

Komplett kod finns på kurssidan.

Renderingens olika steg

1) Rendera från ljuskällan till FBO (med enbart Z-buffer)

2) Rendera med Z-buffern projicerad över scenen från ljuskällan

3) I shaders, använd Z-buffern för skuggtest

1) Rendera från ljuskällan till FBO (med enbart Z-buffer)

```
// Bind the depth FBO
    glBindFramebuffer(GL_FRAMEBUFFER,fboId); //Rendering offscreen

    //Using a simple shader to render to the depthbuffer
    glUseProgram(simpleShader);

    // Modify viewport
    glViewport(0,0,RENDER_WIDTH * SHADOW_MAP_RATIO,RENDER_HEIGHT* SHADOW_MAP_RATIO);

// Clear previous frame values
glClear(GL_DEPTH_BUFFER_BIT);

//Disable color rendering, we only want to write to the Z-Buffer
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);

// Setup the projection and modelview from the light source
setupMatrices(p_light[0],p_light[1],p_light[2],l_light[0],l_light[1],l_light[2]);

// Culling switching, rendering only backface, this is done to avoid self-shadowing
glCullFace(GL_FRONT);
drawObjects();

//Save modelview/projection matrice into the texture matrix, also add a bias
setTextureMatrix();
```

FBO

Enbart Z

Ev culling-trick

Spara matriser för texturprojicering

2) Rendera från kamera

```
// Now rendering from the camera POV, using the FBO to generate shadows
    glBindFramebuffer(GL_FRAMEBUFFER,0);

    glViewport(0,0,RENDER_WIDTH,RENDER_HEIGHT);

    //Enabling color write (previously disabled for light POV z-buffer rendering)
    glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);

    // Clear previous frame values
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //Using the shadow shader
    glUseProgram(shadowShaderId);
    glUniform1i(shadowMapUniform,TEX_UNIT);
    glUniform1i(texunit,TEX_UNIT);
    glActiveTexture(GL_TEXTURE0 + TEX_UNIT);
    glBindTexture(GL_TEXTURE_2D,depthTextureId);

    // Setup the projection and modelview from the camera
    setupMatrices(p_camera[0],p_camera[1],p_camera[2],l_camera[0],l_camera[1],l_camera[2]);

    glCullFace(GL_BACK);
    drawObjects();

    glutSwapBuffers();
```

Ingen FBO - nu är den textur

Shaders och textur

3) Fragment shader

```
uniform sampler2D shadowMap;

in vec4 shadowCoord; // Surface position in light source coordinates
out vec4 fragColor;

void main()
{
   // Perform perspective division to get the actual texture position
   vec4 shadowCoordinateWdivide = shadowCoord / shadowCoord.w ;

   // Used to lower moire' pattern and self-shadowing
   // The optimal value here depends on Z buffer resolution.
   shadowCoordinateWdivide.z += 0.002; // 0.0005;

   // Look up the depth value
   float distanceFromLight = texture(shadowMap, shadowCoordinateWdivide.st).z;

   // Compare
   float shadow = 1.0;
   if (shadowCoord.w > 0.0)
      if (distanceFromLight < shadowCoordinateWdivide.z)
         shadow = 0.5;
   fragColor = shadow * gl_Color;
}
```

Texturprojektion, samma som tidigare

Men nu hämtar vi Z-värdet...

...och jämför med avståndet till kameran, som vi redan har

Snyggt?

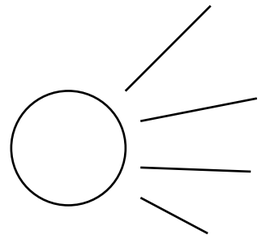
Nja, det var ju hyfsade skuggor - bitvis. Kastade skuggor blir fina!

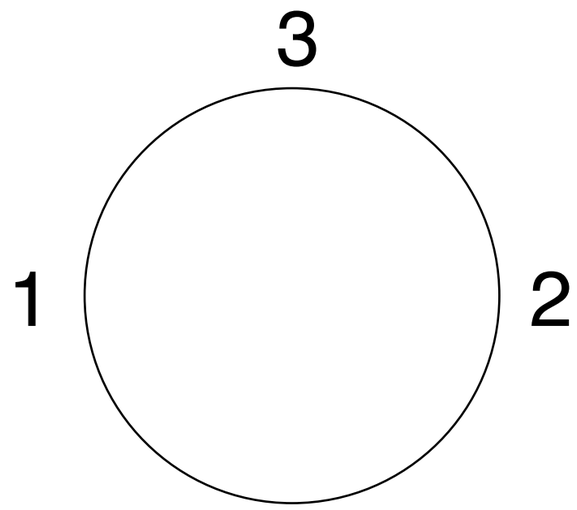
Tre problemområden:

1) Belysta områden.

2) Attached shadow, objektens baksidor.

3) Gränsområdet mellan dessa två.



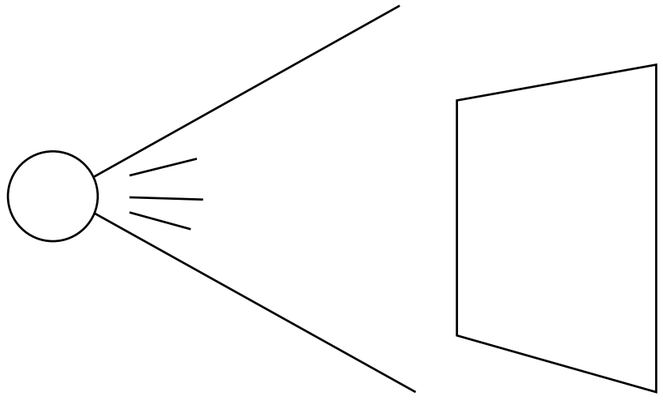


Problem med 1) kan elimineras av culling...
men då flyttas problemen till 2).
3) är det värsta, beror på upplösningen

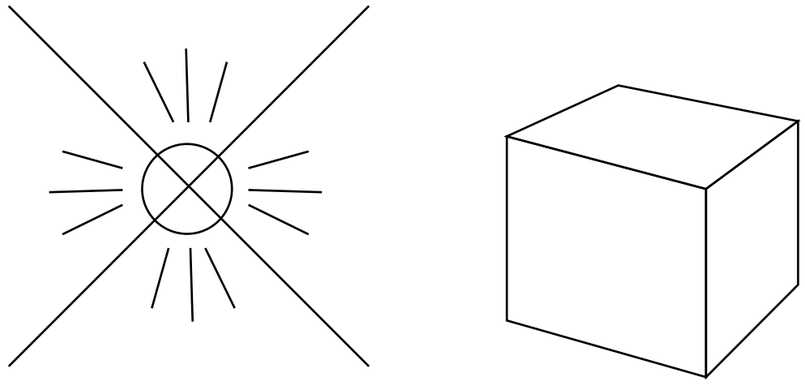
Omnidirectional shadow map

Problem: Begränsat frustum.

Utvidgning: Rendera flera riktningar. I princip en *cube map*.



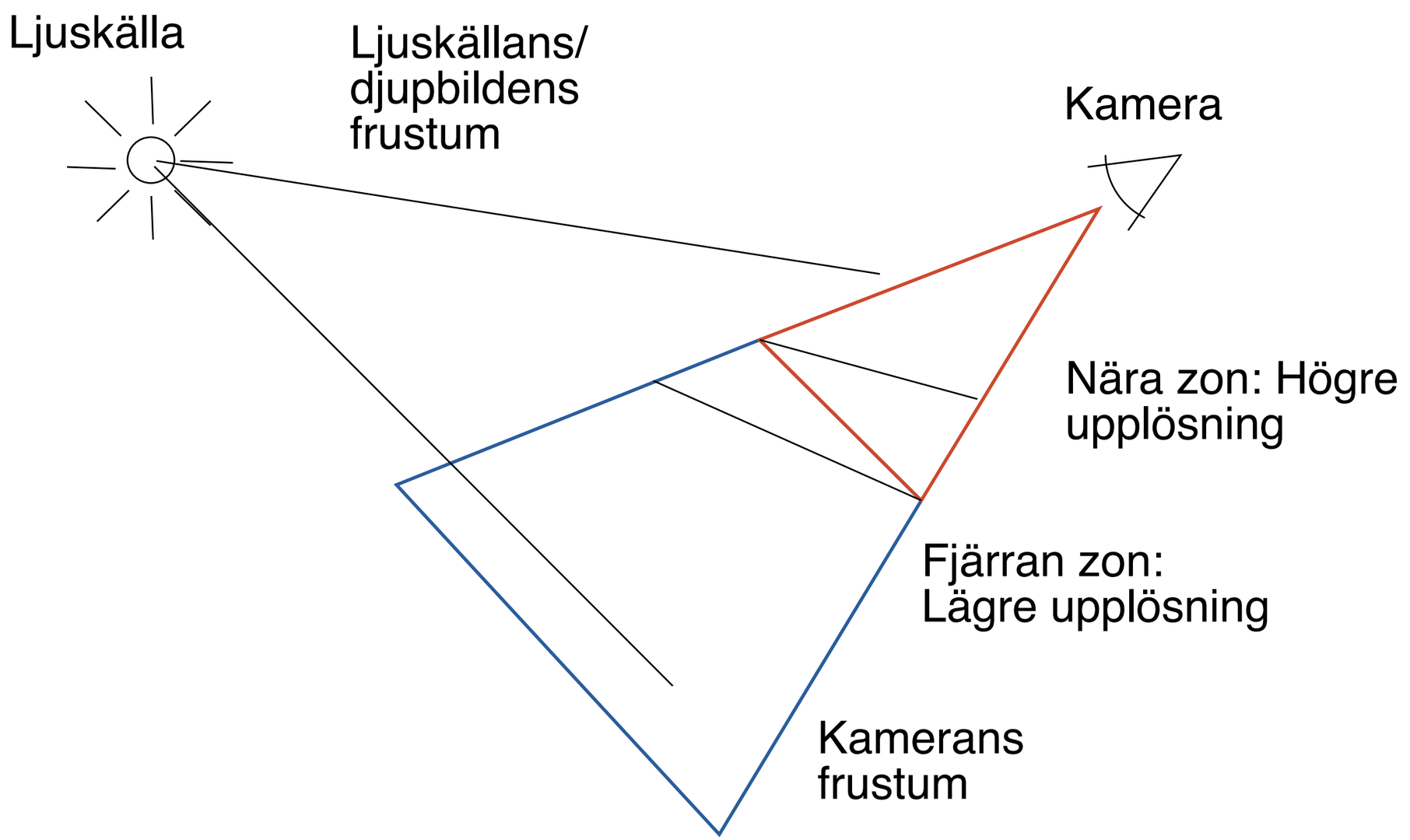
Ett frustum, en shadow map

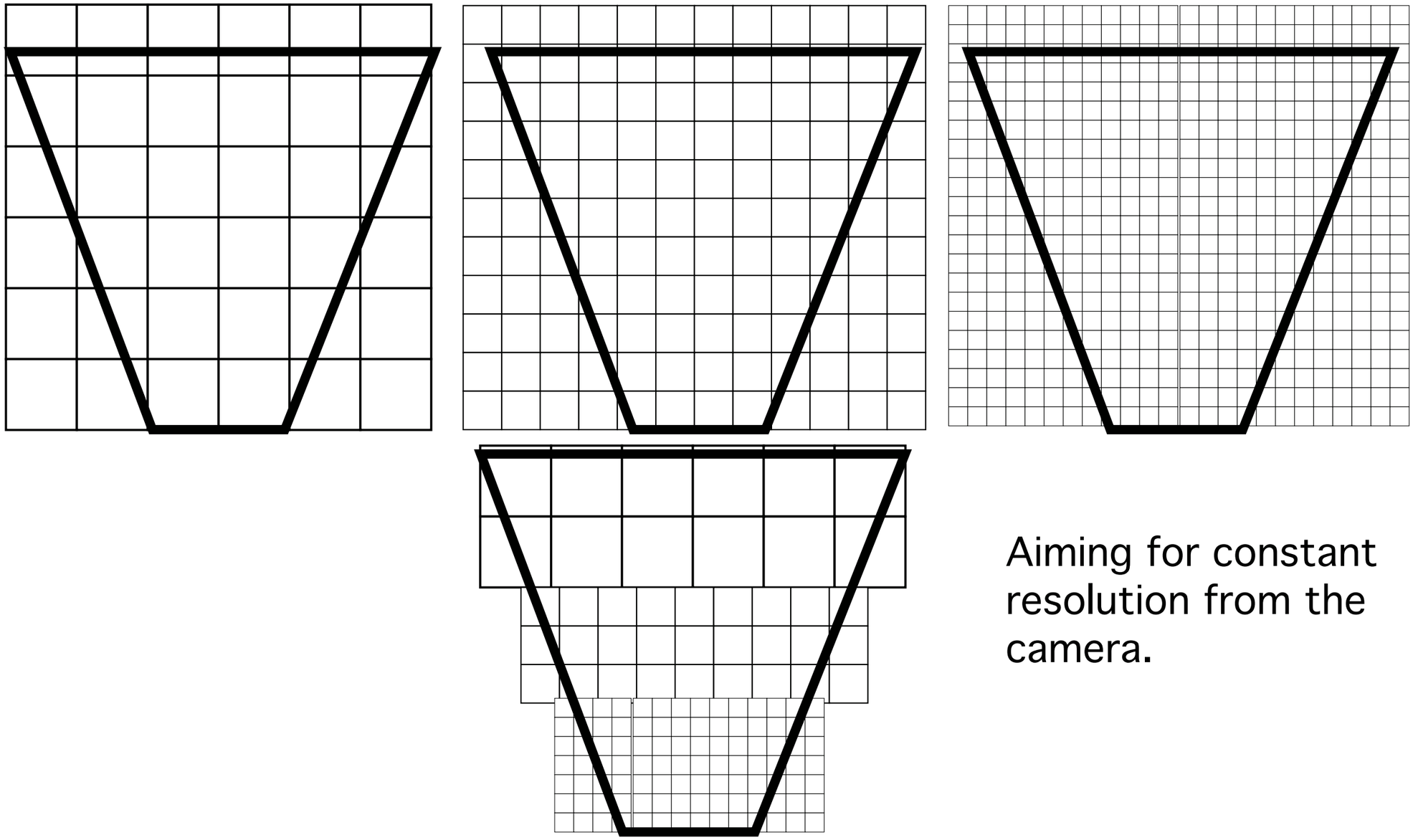


Flera frustum, en kub av shadow maps

Cascaded shadow maps

Löser/förbättrar problem 2! Dela upp viewing frustum i flera zoner, rendera till olika Z-buffrar, med olika upplösning, i de olika zonerna.


Ljuskälla
Ljuskällans/
djupbildens
frustum
Kamera
Nära zon: Högre
upplösning
Fjärran zon:
Lägre upplösning
Kamerans
frustum



Aiming for constant resolution from the camera.

Vi återkommer till shadow mapping straxt...