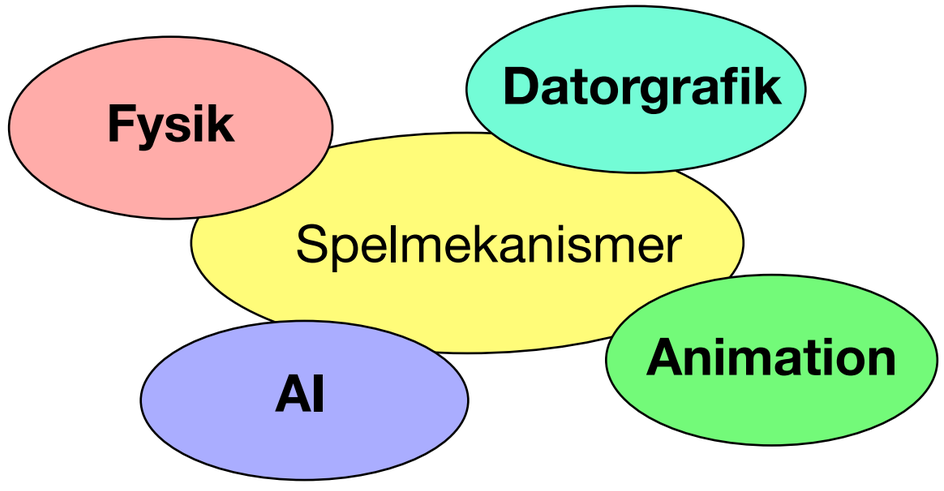
TSBK 03
Teknik för avancerade datorspel

Ingemar Ragnemalm, ISY



Föreläsning 3

"Konventionell" skuggenerering

- Skuggor, definitioner
  - Plana skuggor
  - Skuggmappning
  - Skuggvolymer
  - Mjuka skuggor
- Ambient occlusion

Skuggor är viktiga eftersom:

- bidrar med realism
- viktiga “depth cues” som gör det lättare att förstå scenen (speciellt för flygande föremål)

men 3D-API’erna löser inte problemet åt oss automatiskt!

Skuggor

Vi vet redan att vi kan göra skuggor på flera sätt:

- Strålföljning
- Radiosity
- Light mapping baserad på dessa

men ingen av dem ger dynamiska skuggor!

(Oräknat strålföljning i realtid - helst med RTX.)

Tar strålföljningsbaserade metoder över helt?

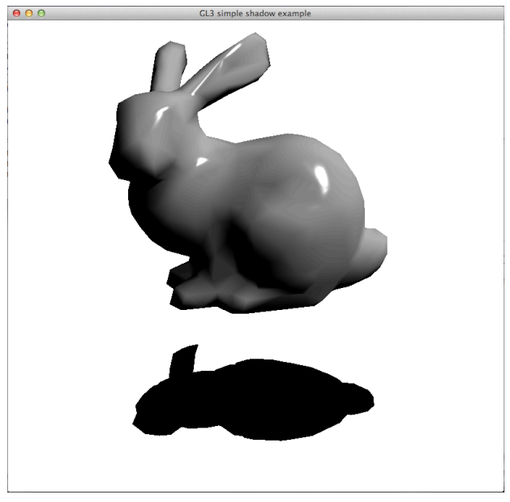
100% skifte? Inte än!

- RTX eller billigare GTX? RTX verkar ha tagit över helt!
- Low-end (telefoner mm)?
- Andra prioriteringar kräver beräkningskraften?
- Hybridsystem, utnyttjar RT för delar av scenen.

En billig metod:

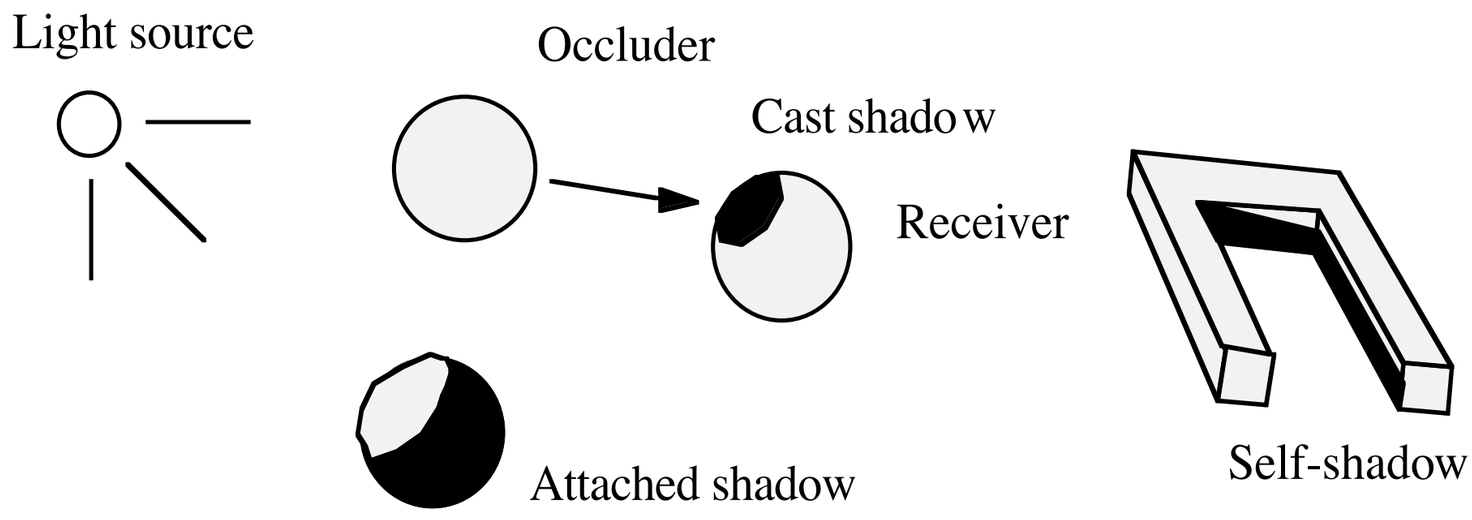
Rita tillplattad modell på ytan.

Hyfsat på ”oändlig” yta eller med stencilbuffer.



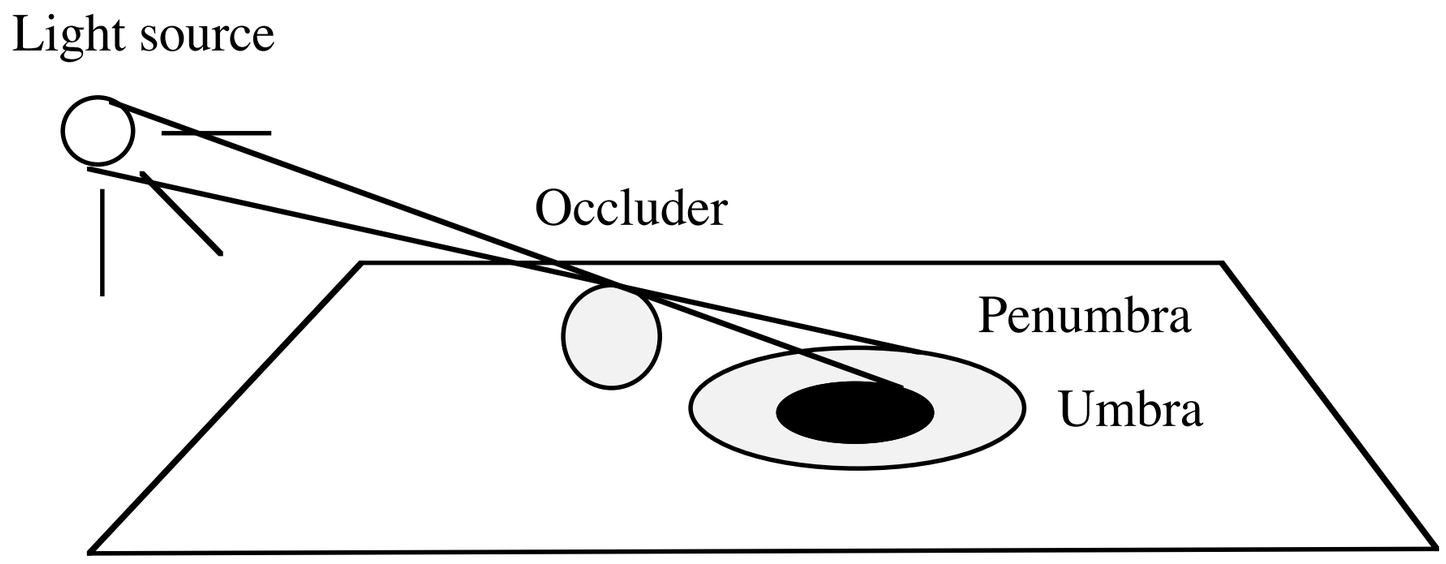
Definitioner

Occluder
Receiver
Attached shadow
Self-shadow



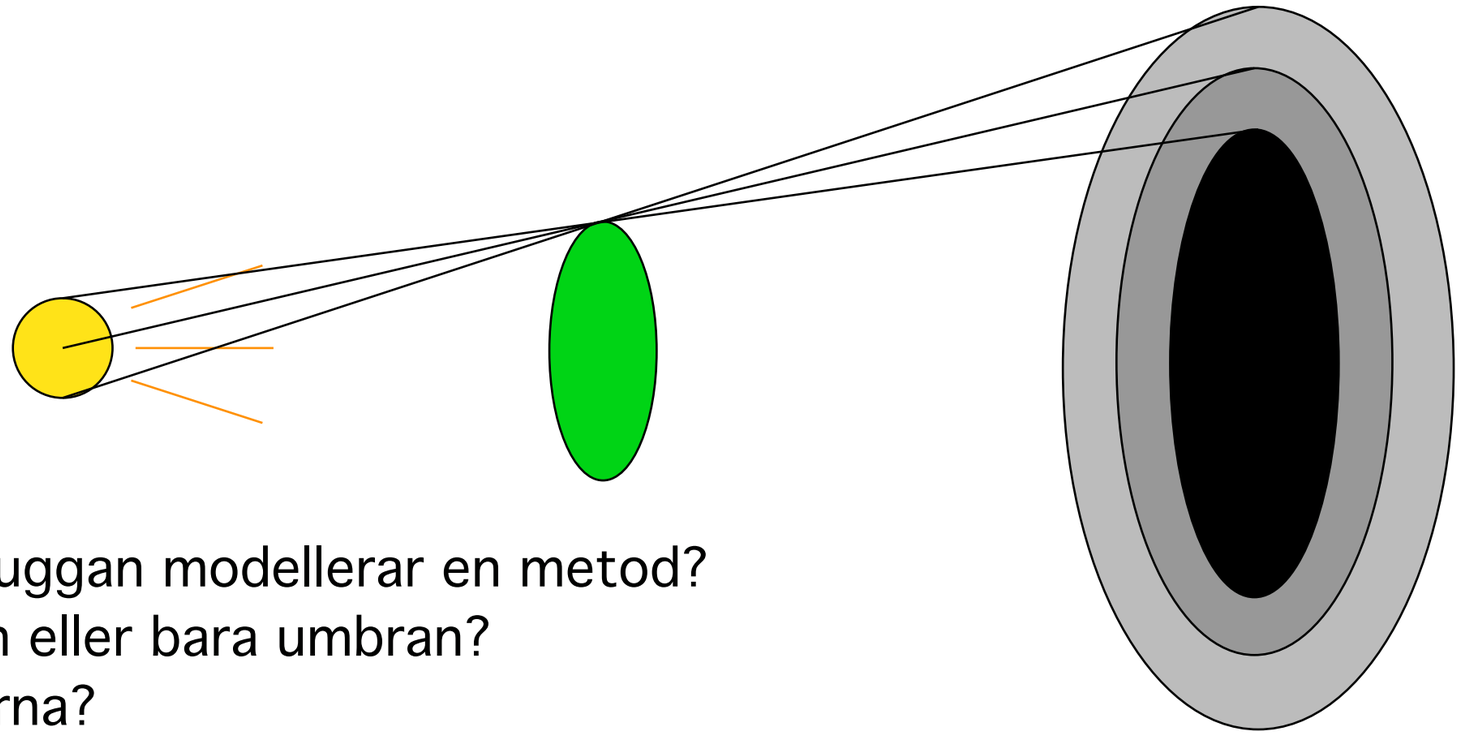
Definitioner

Umbra
Penumbra (yttre & inre)



Penumbran

Inre penumbra - skugga om punktkälla
Yttre penumbra - belyst om punktkälla



Vilka delar av skuggan modellerar en metod?
Finns penumbran eller bara umbran?
Finns båda zonerna?

Tre huvudsakliga sorters skuggor:

• Attached shadow. Detta klarar vi redan med den vanligaste belysningsmodellen.

Men det gäller inte för:

• Cast shadow, skugga från objekt till objekt.

• Self-shadowing.

Vi behöver effektiva metoder (dvs realtid) för detta som ger bra resultat.

Dynamiska skuggor

Skuggmetoder med låga prestandakrav.

Metoder för dynamiska skuggor:

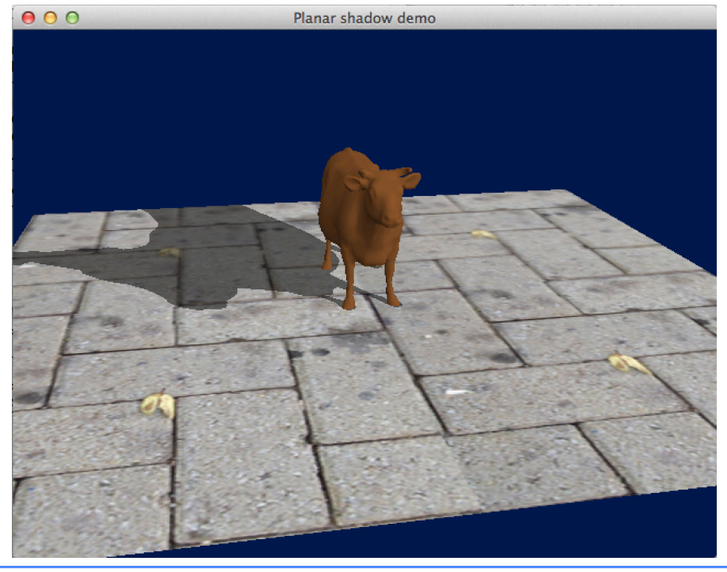
- Projektion på plana ytor
  - Shadow maps
  - Shadow volumes

Projektion på plan yta

Projicera objektet på en yta!

Projektionsmatris multipliceras in i stil med rotation runt godtycklig axel.

Objektet renderas utan textur, i önskad skuggas färg, blendas med bakgrunden.



Projektion på plan yta

Delproblem att lösa:

- Utför projektionen av objektet till vald yta
- Maska ritandet till denna yta med stencilbuffern
- Rita objektet på ytan med transparens

Projektionsmatrisen

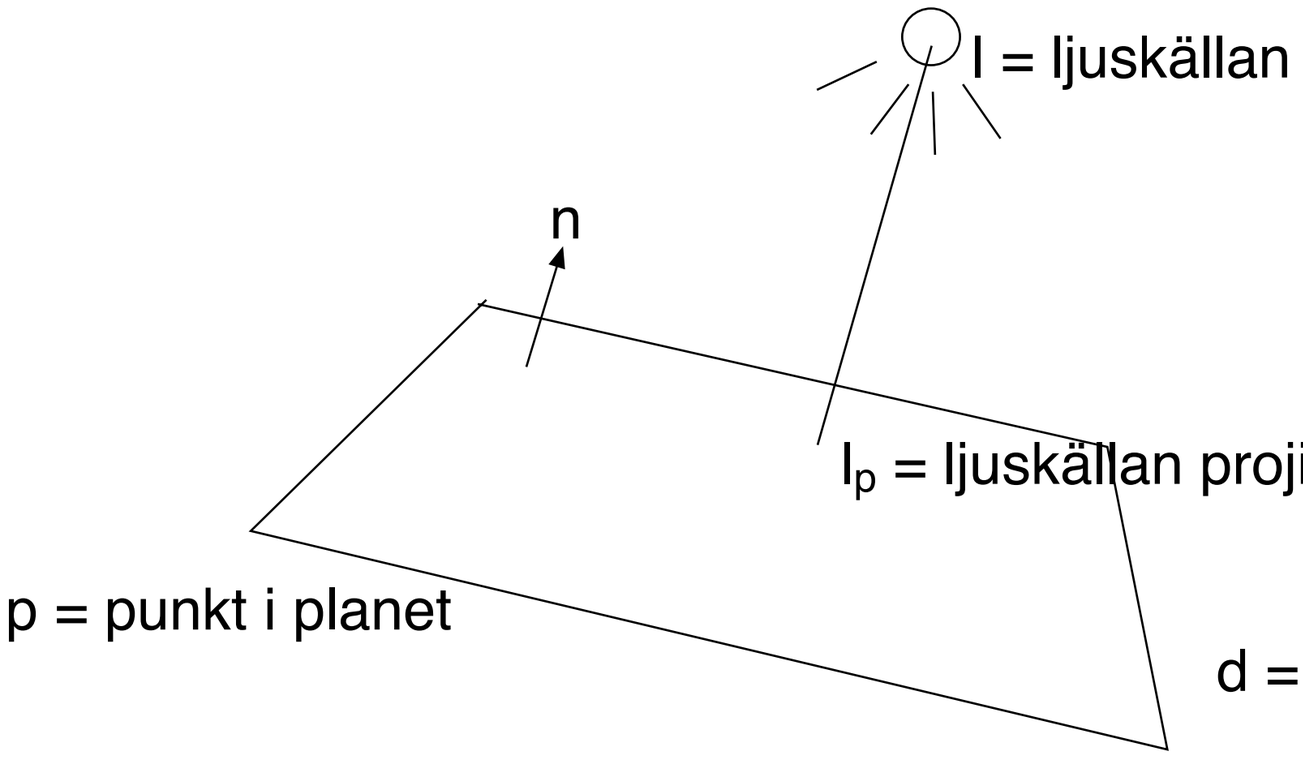
Enkla projektionmatriser längs Z:

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & f \end{bmatrix}$$

Rotera och translatera så det stämmer med önskat plan.

Höger matris projicerar på Z=0.
Vänster har ljuskällan i origo.

Variabler i projektionen



$d = -p \cdot n$

$l_p = l - n(n \cdot l + d)$

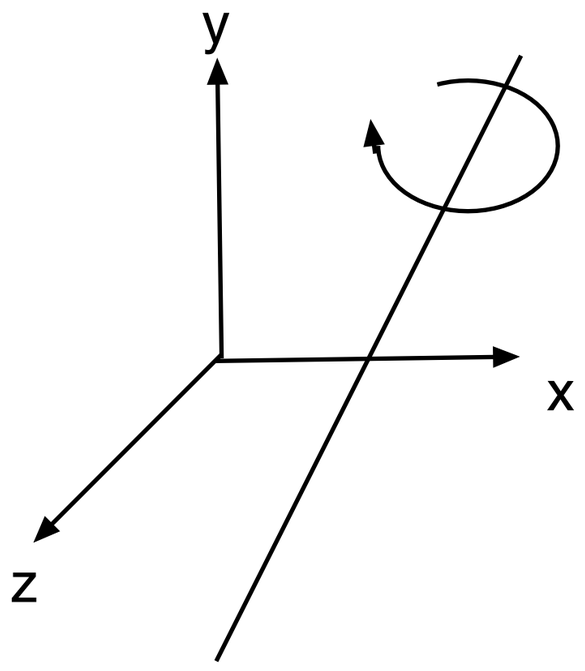
Projektionstransform

Samma princip som många liknande fall i grundkursen.

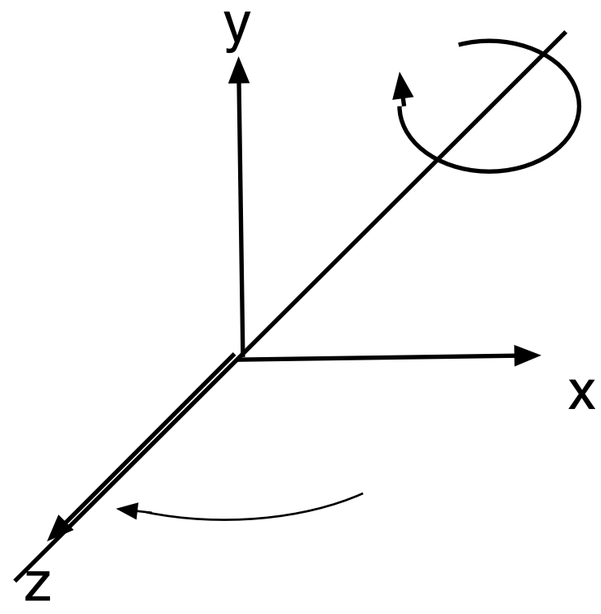
Låt l_p vara ljuskällan projicerad på planet.

- Translatera l_p till origo
- Rotera normalvektorn längs Z
- Projicera
- Rotera tillbaka
- Translatera tillbaka

Rotation runt godtycklig axel



Transform to align axis with the Z axis, rotate, and transform back.



Total transformation:

$$R(\theta) = T(p_1) * R^T * R_z(\theta) * R * T(-p_1)$$

Kod för projektion

```
float d = - p * n;
float f = n * l + d; // Distance light source to plane = focal distance
vec3 lp = l - n * (n * l + d); // lp = l - n(n dot l + d)

mat4 toz = AxisToZ(n);
mat4 fromz = Transpose(toz);
mat4 shadowProjectionMatrix = CreateShadowProjectionMatrix(f);

mat4 sm = T(lp) * fromz * shadowProjectionMatrix * toz * T(-lp);
```

Total transformation:

$$Proj = T(lp) * R^T * P * R * T(-lp)$$

Rita golvet i stencil, framebuffern och Z-buffer

```
// Inline geometry
GLfloat floorVertices[] =
     {100.0f, 0,-100.0f,
     -100.0f, 0,-100.0f,
     -100.0f, 0,100.0f,
     100.0f, 0,100.0f};
GLfloat floorTex[] = { 4.0f, 4.0f, 0.0f, 4.0f,
                       0.0f, 0.0f, 4.0f, 0.0f};
GLuint floorIndices[] = {0, 1, 2, 0, 2, 3};
```

Inlinegeometri laddas upp
(till ”model” i LittleOBJLoader)

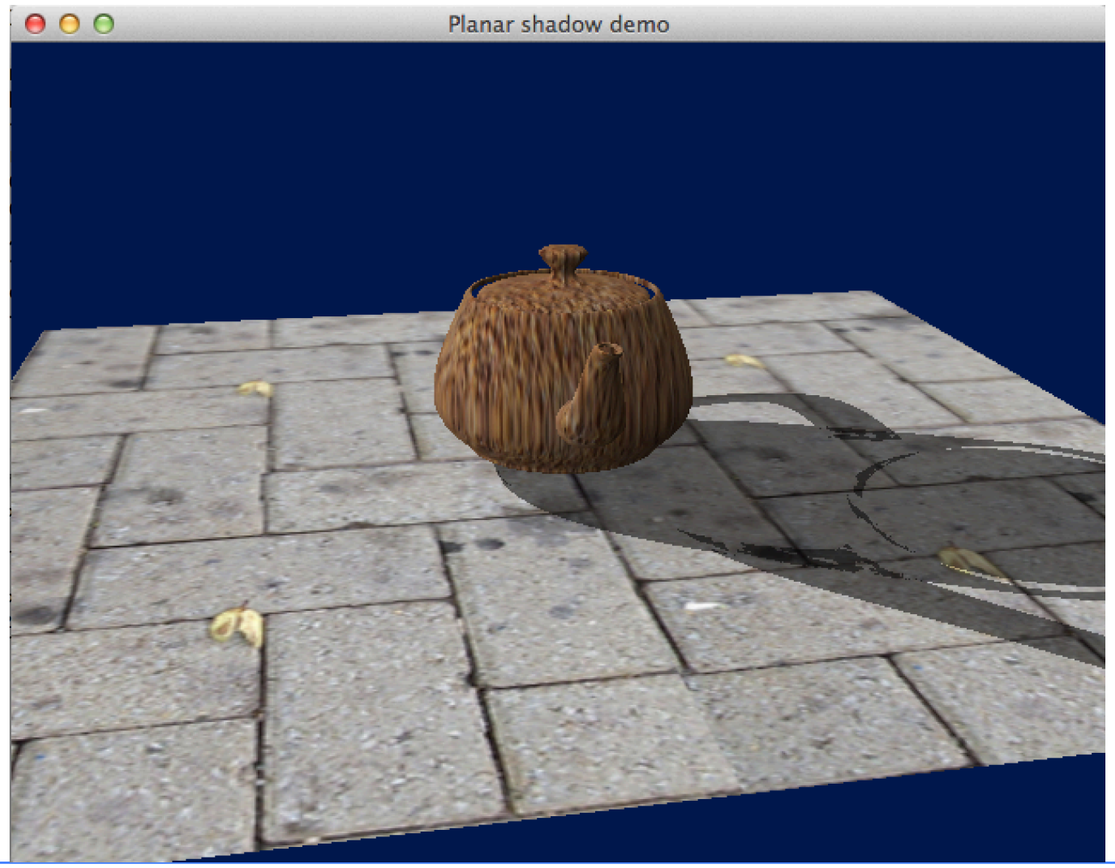
Skriv 1 om pass!
Annars keep!
Vi vill inte rita i skymd yta!

```
glStencilFunc(GL_ALWAYS, 1, 0xFFFFFFFF);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE); // 1 if pass
// Draw the floor
glBindTexture(GL_TEXTURE_2D, TextureArray[0]);
// Upload any shader variables here
DrawModel(floorModel);
```

Rita på vanligt sätt

Multipel blending

Ett fall för stencilbuffern!



Multipel blending

Vi använde bara stencilbuffern som binär mask. Den anger om vi skrivit en bakgrund i den eller ej. Men vi vill ju helst testa om vi skrivit skugga också!

Och detta visar sig vara ganska lätt!

Vi hade

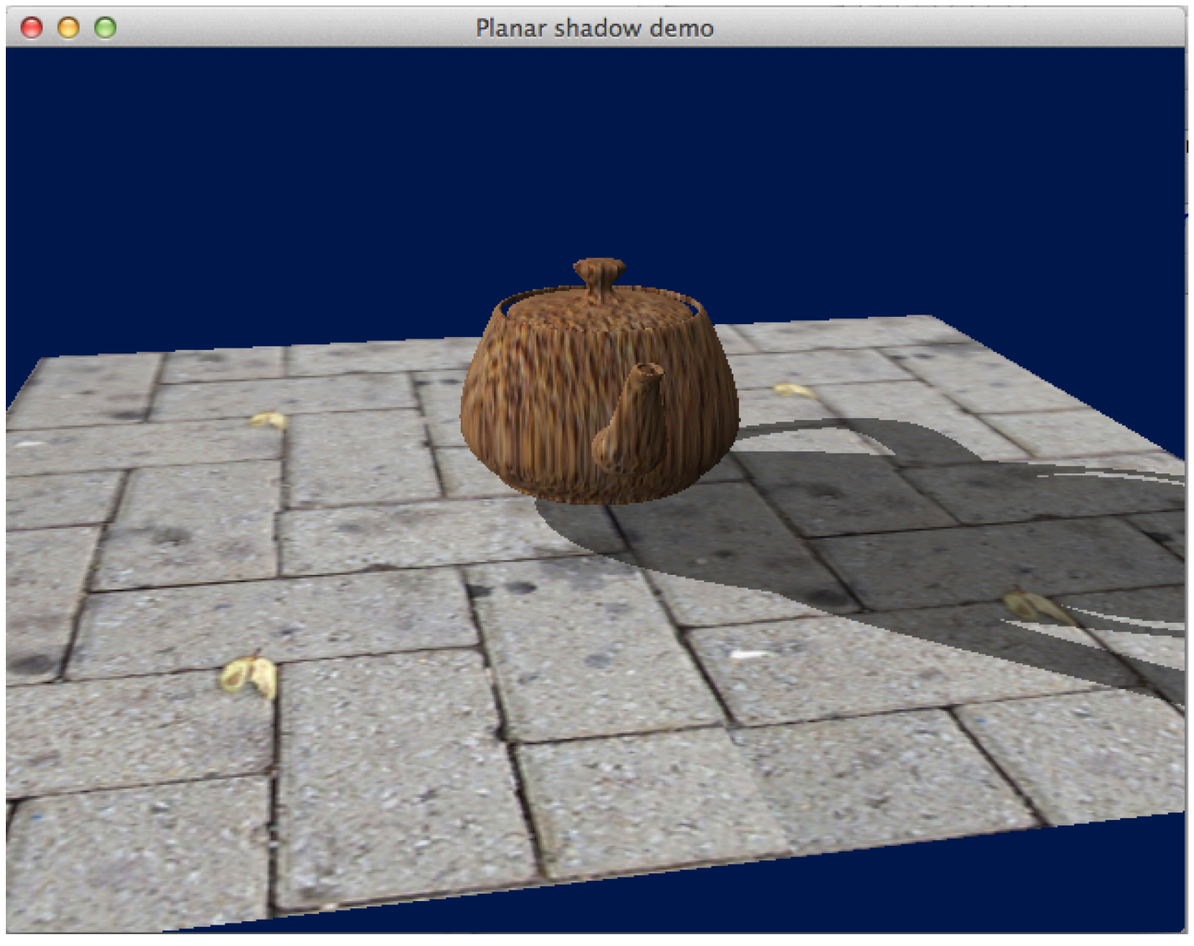
```
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
```

men vad sägs om denna?

```
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
```

Dvs rita om stencil = 1 (enl glStencilOp) men öka med 1 om det går igenom!

Multipel blending korrigerat, resultat



Projektion på plana ytor

- Enkelt att göra med projektionsmatris och stencilbuffern

- Bara plana ytor! Bökigt och långsamt för flera ytor, kräver att stencilbuffern raderas och ritas om för varje yta!

- Problem vid blendning av komplexa objekt

Bättre kan vi! Shadow maps och shadow volumes ger bättre resultat!

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.

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!

Beslut baserade på Z-buffern

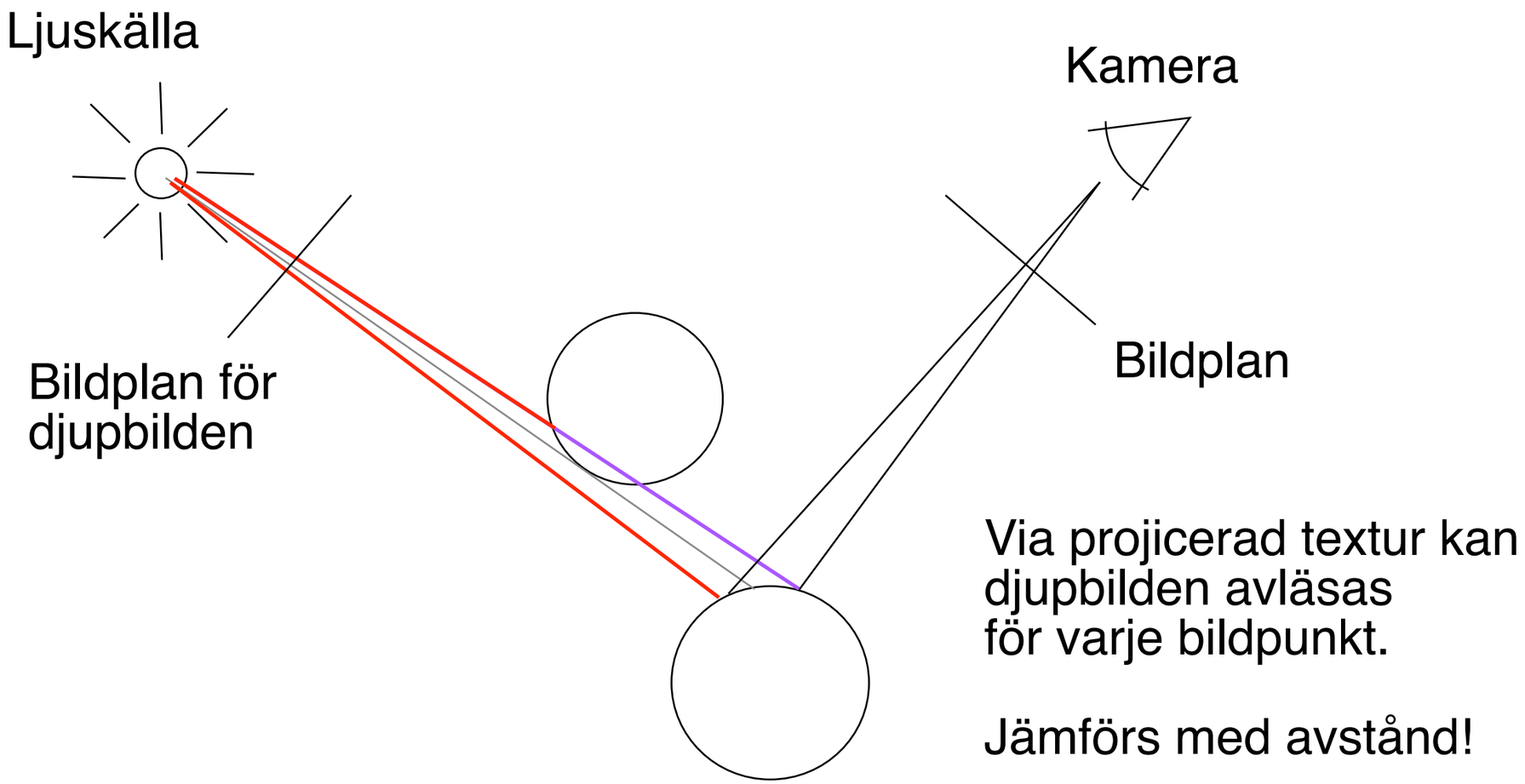
Z-buffern är en “djuptextur”. När den projiceras över scenen så nås varje fragmentberäkning av ett Z-värde.

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.

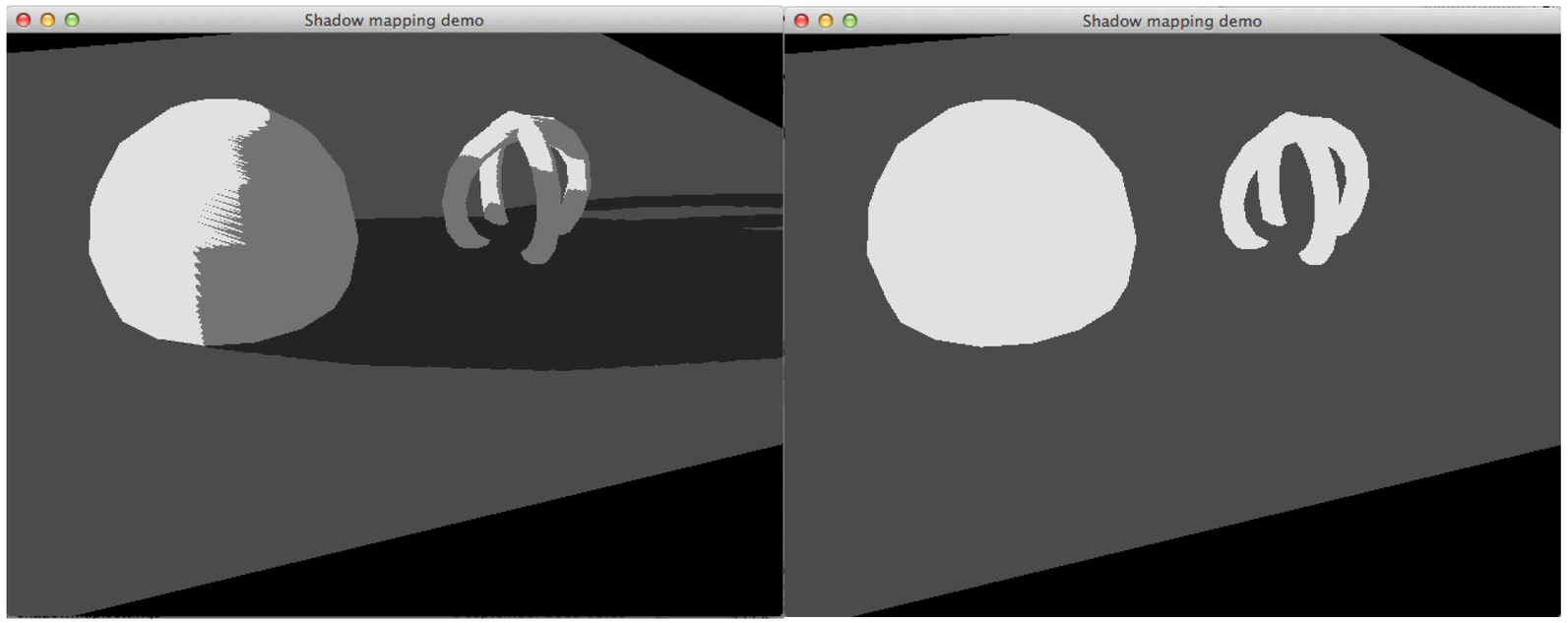
Shadow maps - princip



Via projicerad textur kan djupbilden avläsas för varje bildpunkt.

Jämförs med avstånd!

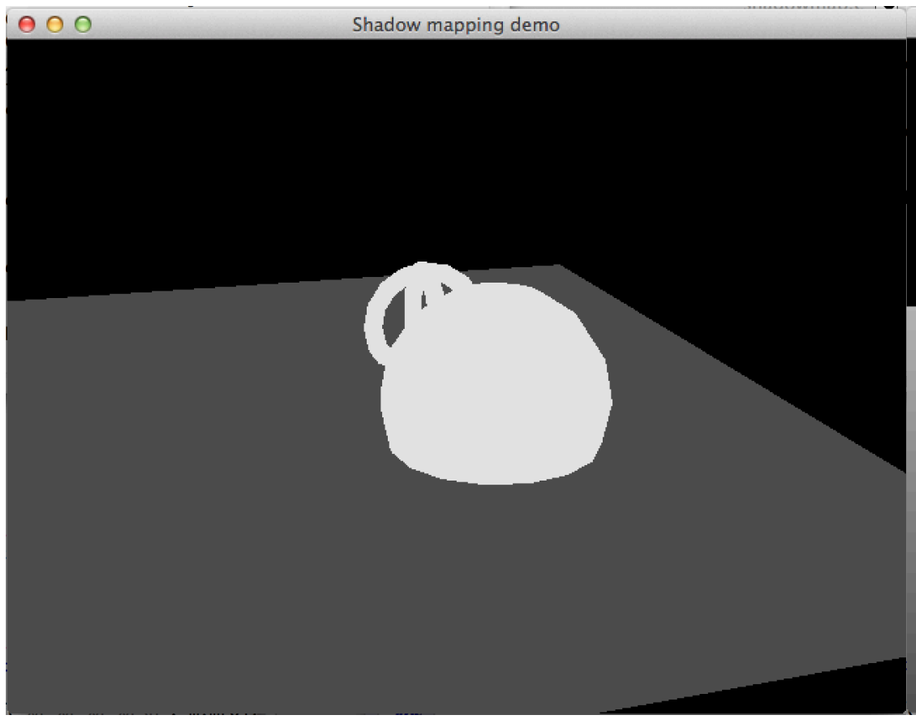
Shadow maps - exempel

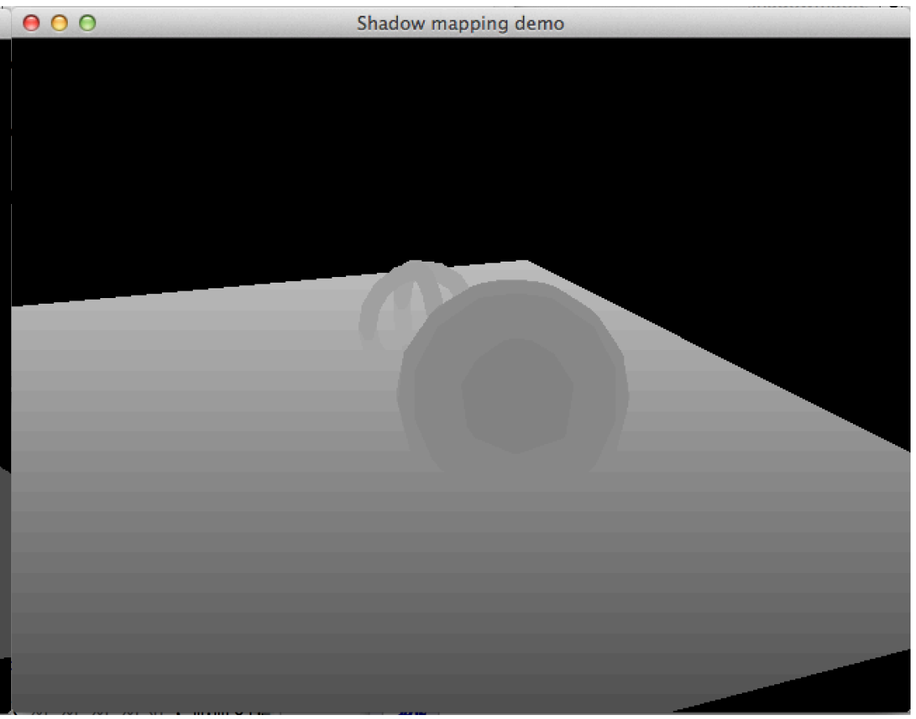


Med skuggor Utan skuggor

Shadow maps - exempel

Rendering av shadow map



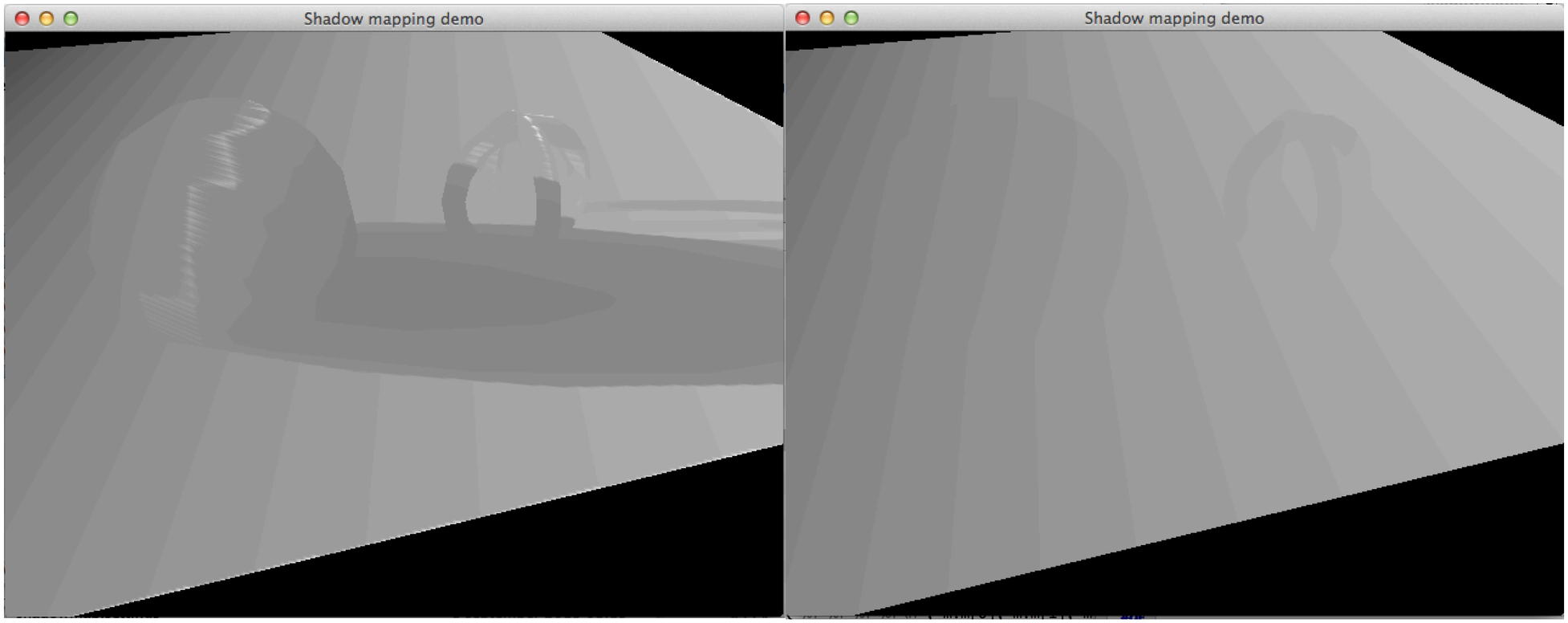


Sedd från ljuskällan

Z-buffer från ljuskällan

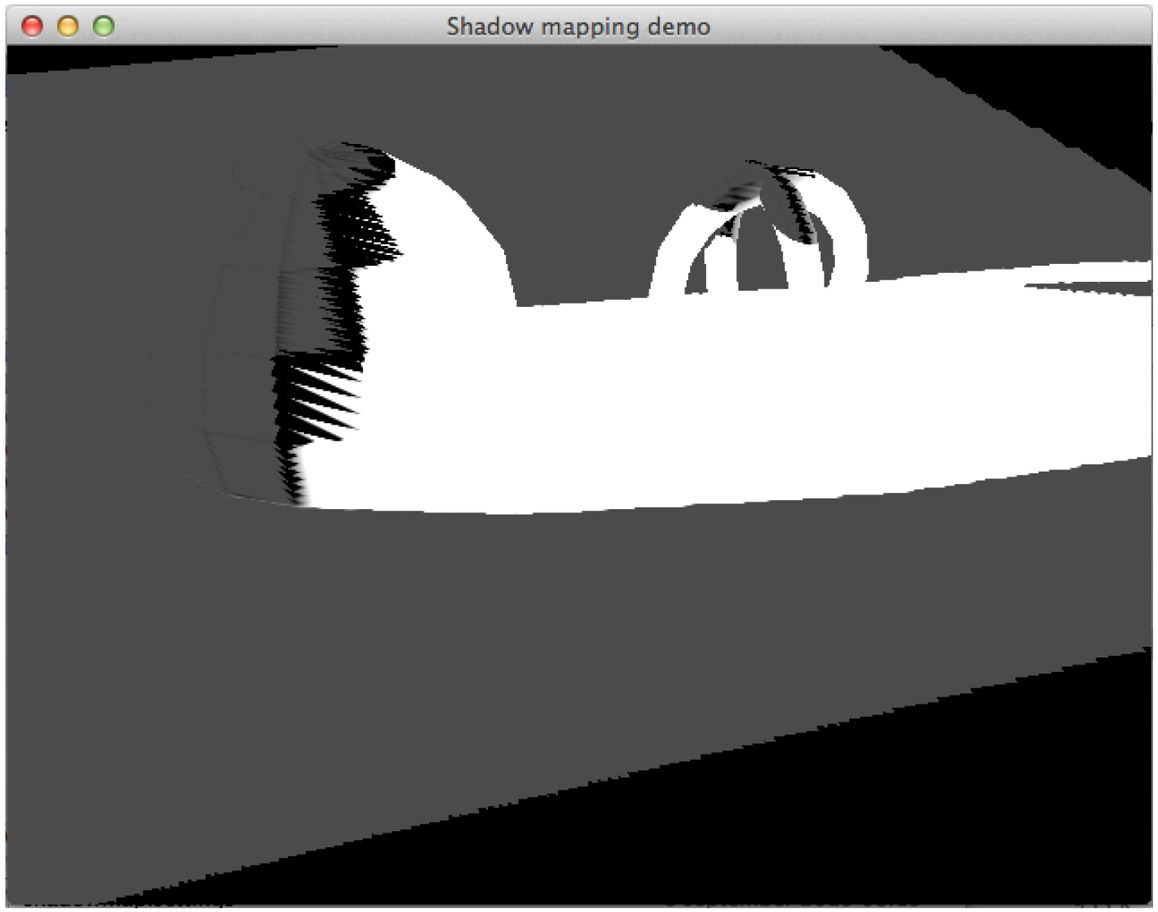
Shadow maps - exempel

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

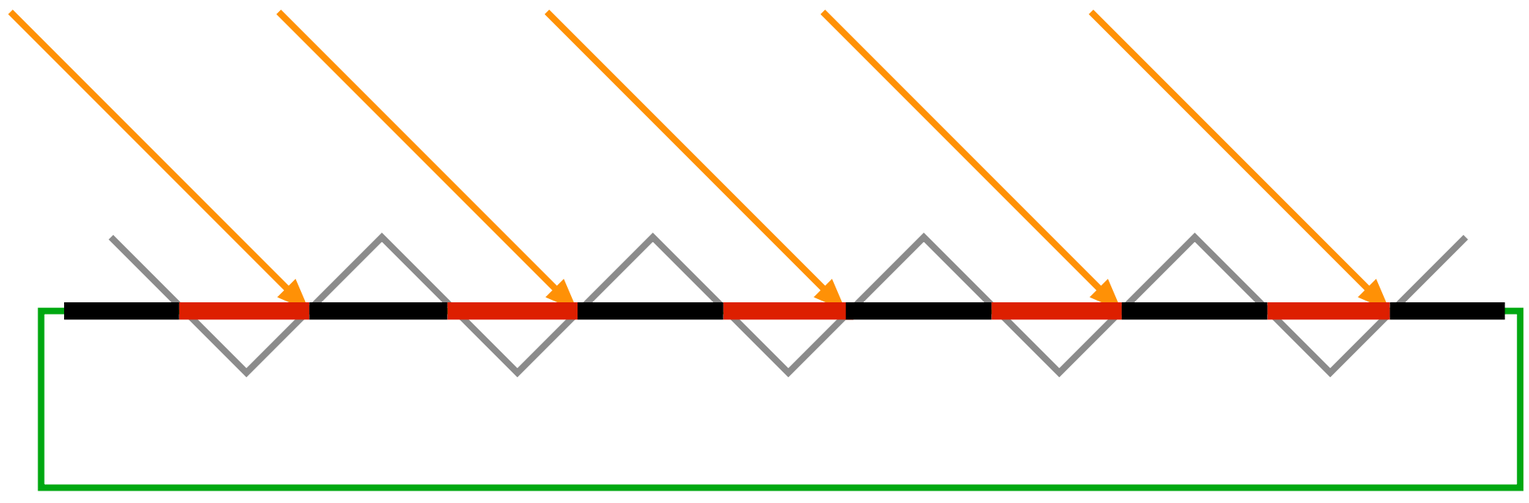
Problem 2: Shadow buffer kräver hög upplösning för att inte ge kantartefakter

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.

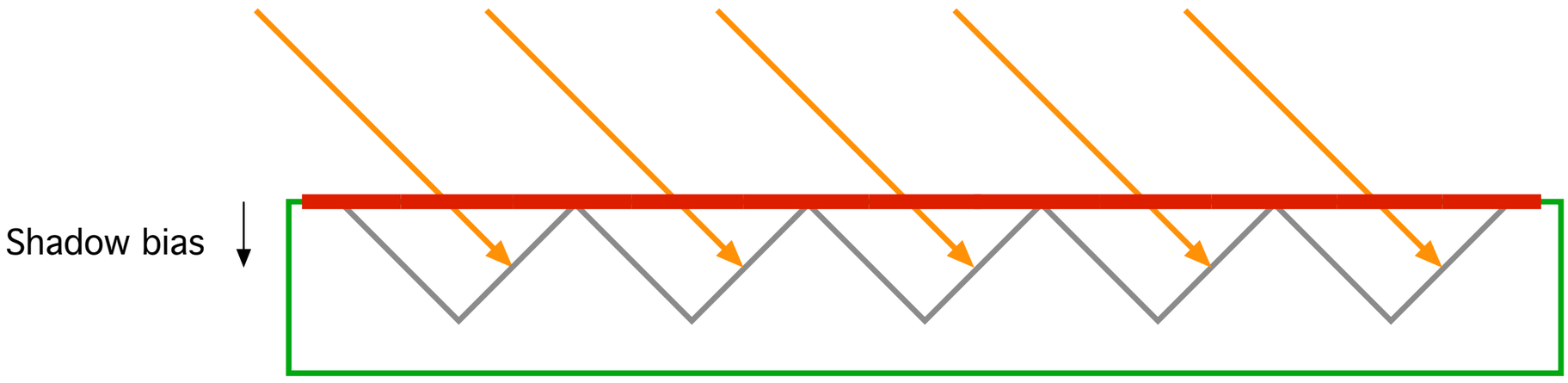
Bild baserad på bild från LearnOpenGL

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.

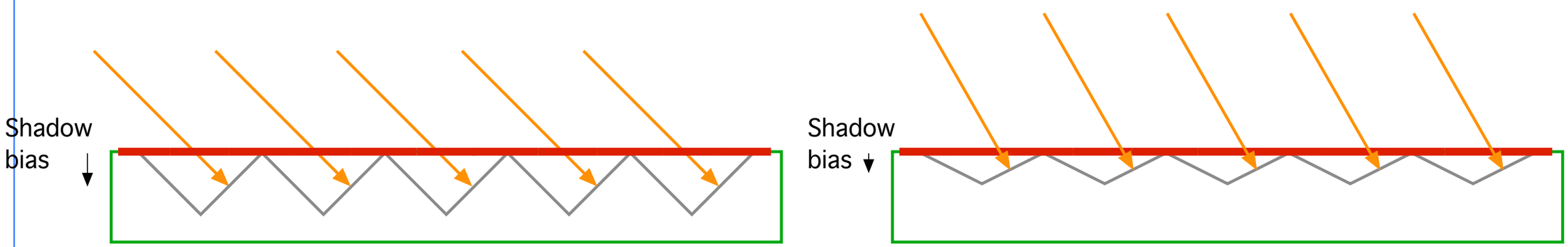
Bild baserad på bild från LearnOpenGL

Lutningsberoende shadow bias

Störst problem i brant lutning.

Ha olika mycket shadow bias beroende på lutning!

Minskar Peter Panning

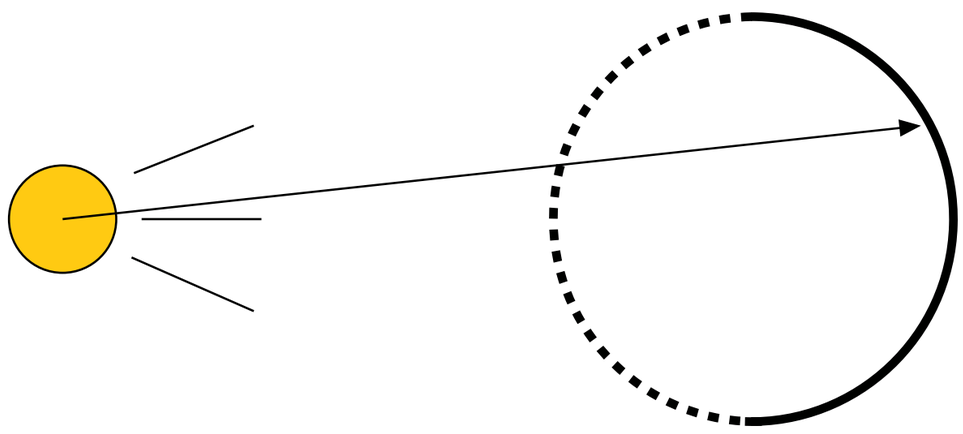


Front-face culling

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 Sanglard,
förenklat, generaliserat.

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. Ger avstånd i z.

Men nu hämtar vi Z-värdet...

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

Snyggt?

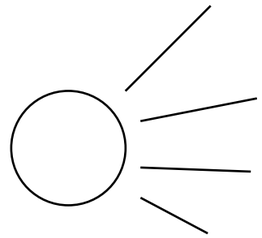
Nja, det var ju hyfsade skuggor - bitvis. Kastade skuggor blir fina!

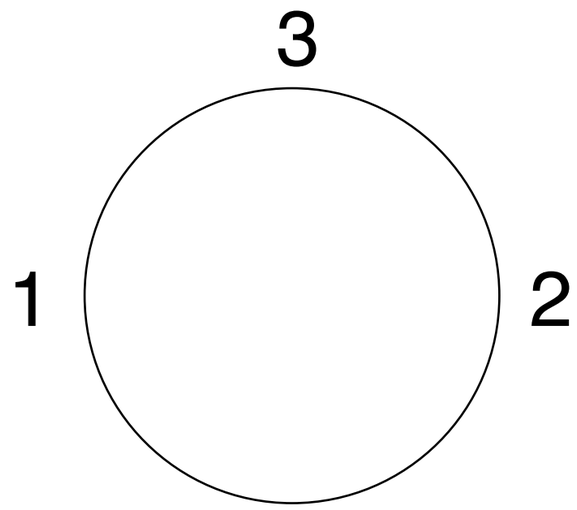
Tre problemområden:

1) Belysta områden.

2) Attached shadow, objektens baksidor.

3) Gränsområdet mellan dessa två.



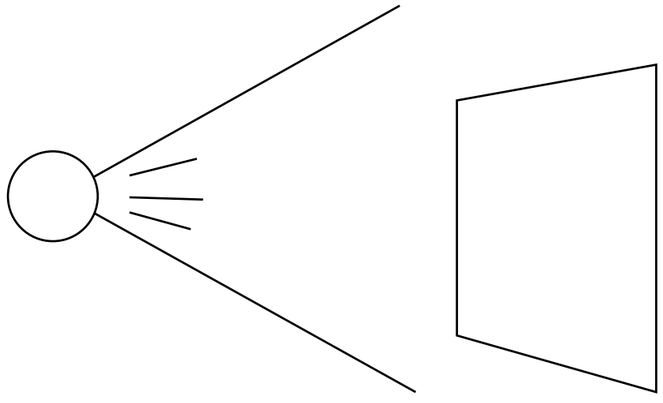


Problem med 1) kan elimineras av culling...
men då flyttas problemen till 2).
3) är det värsta, beror på upplösningen

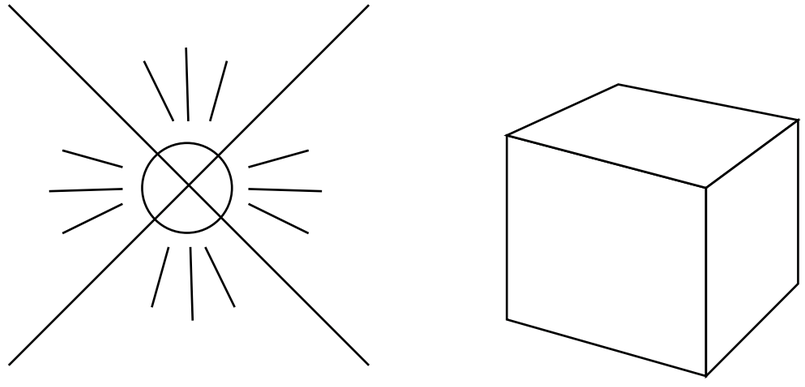
Omnidirectional shadow map

Problem: Begränsat frustum.

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



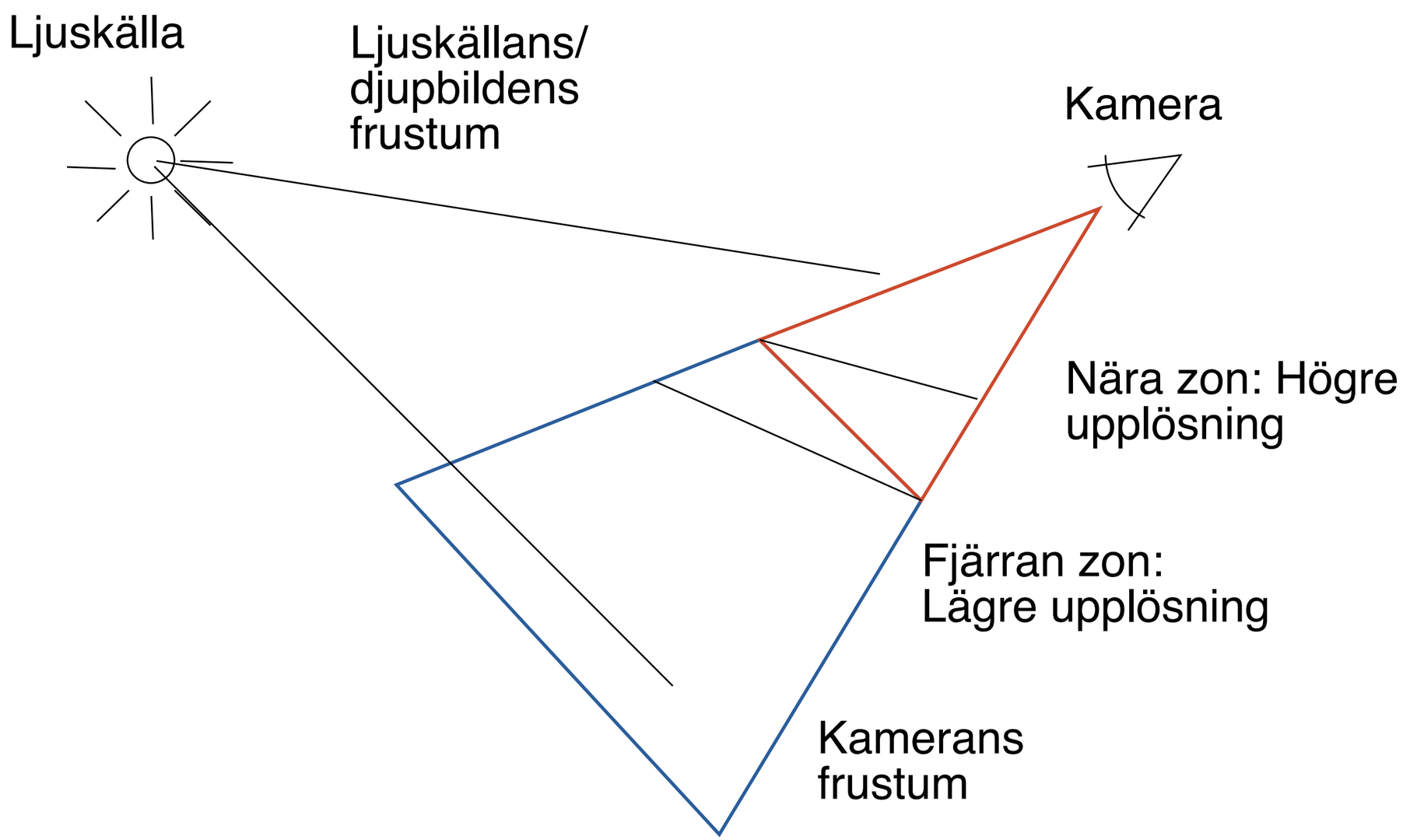
Ett frustum, en shadow map

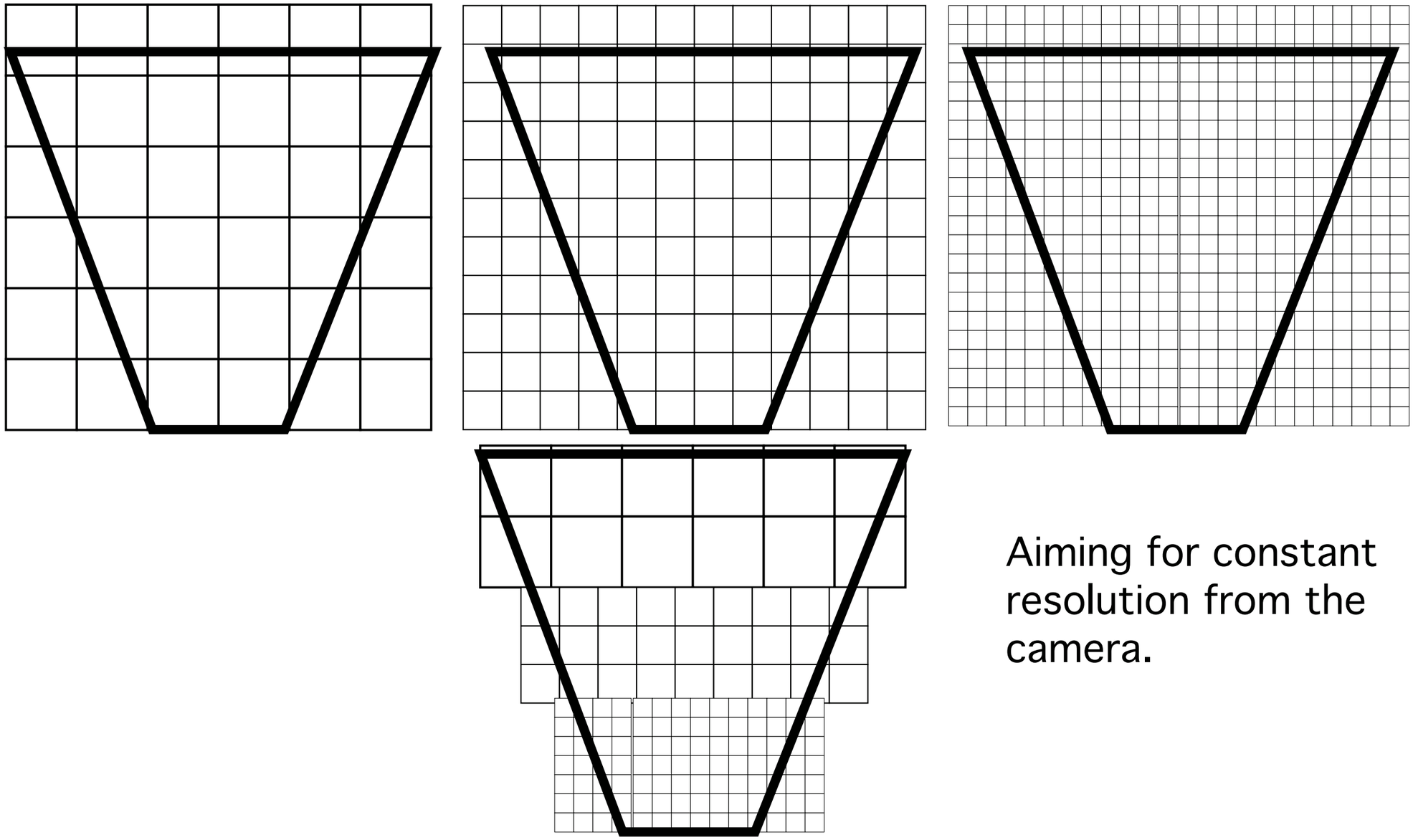


Flera frustum, en kub av shadow maps

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...

Shadow volumes
(Stencil shadows)

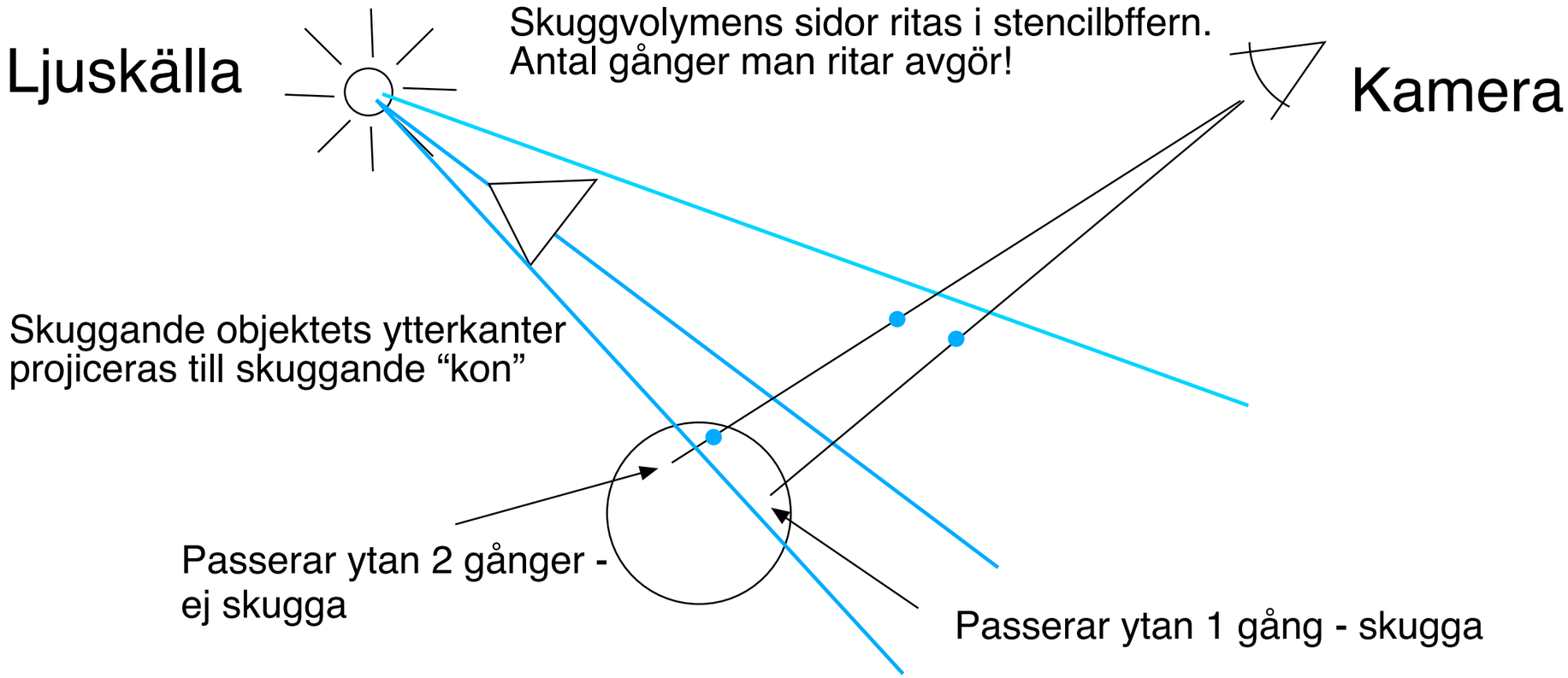
En tredje skuggningsmetod.

Idé: “projicera” skuggande objekt till kantytor på en volym. Med hjälp av stencilbuffern kan vi avgöra om en viss punkt är inom eller utanför volymen.

Fördel: Undviker upplösningsberoendet som shadow maps har.

Shadow volumes

Princip

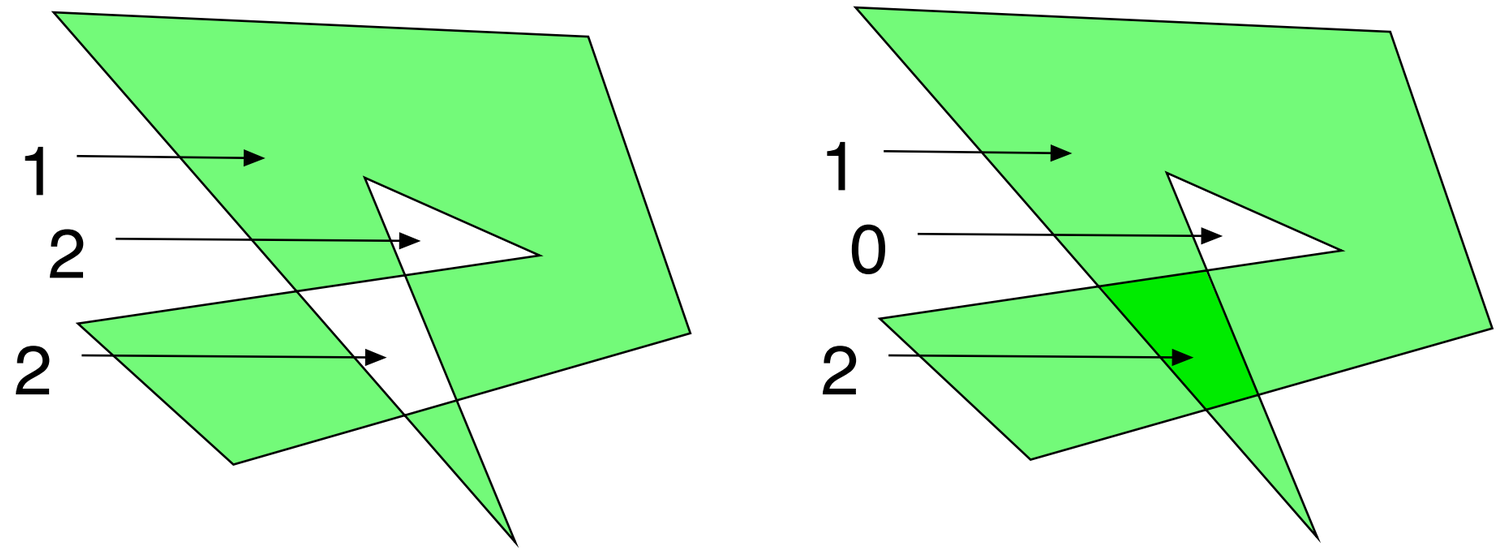


Algoritm

- Bilda polygoner som utgör sidorna på skuggvolymen
- Initiera stencilbuffern till 1 om kameran är i skuggvolymen, annars 0
- Rita alla framsidor i stencilbuffern, med inkrement
- Rita alla baksidor i stencilbuffern, med dekrement
- Rita scenen
- Rita skuggor i alla pixlar där stencilbuffern > 0

Non-zero winding number

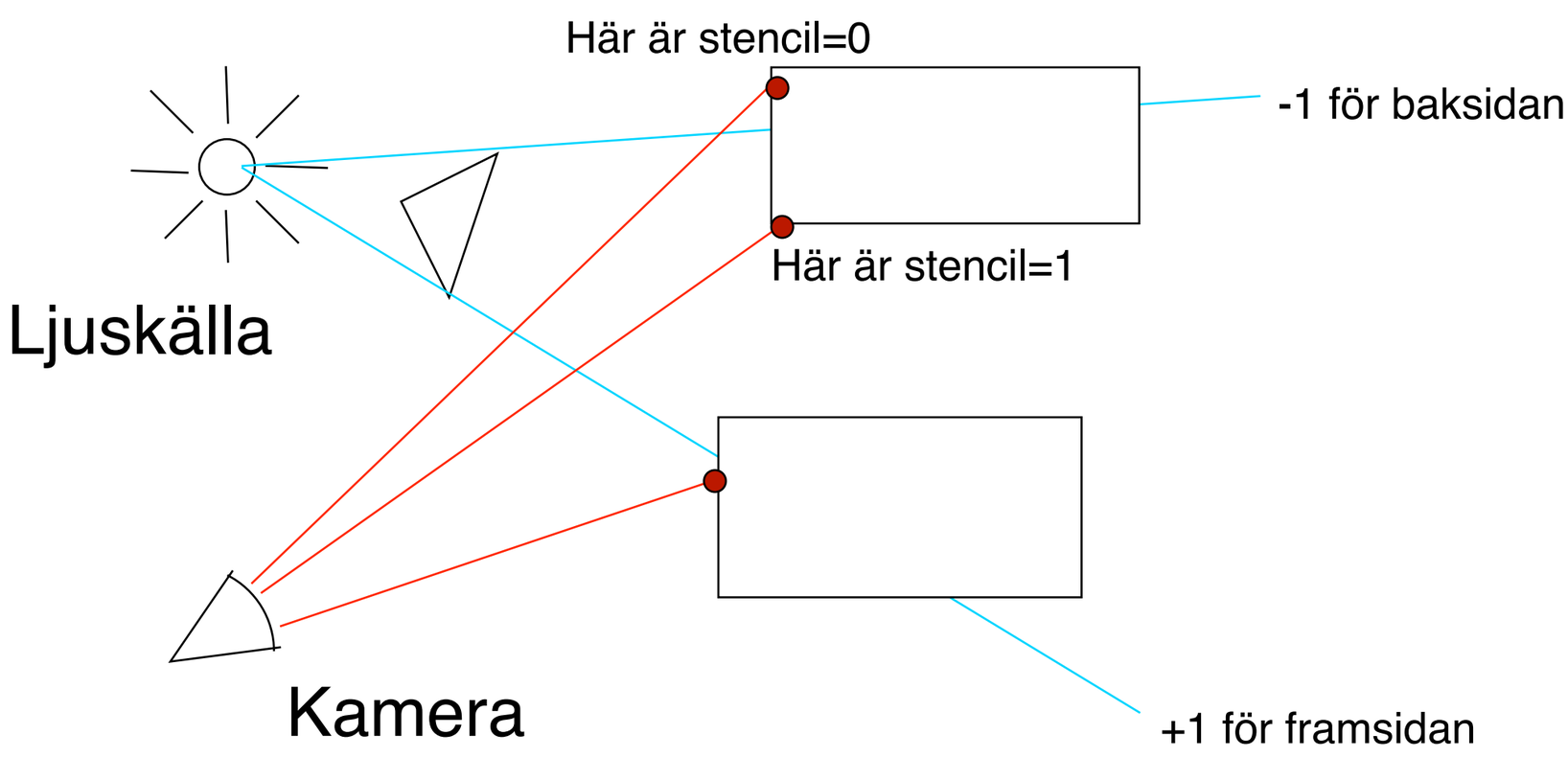
Shadow volumes är samma princip i 3D!



Odd-even rule, udda antal kanter är “innanför” (fel)

Non-zero winding number rule, summan av “upp-kanter” och “ner-kanter” är noll när vi är utanför (rätt)

Stencilbufferns uppdatering och beslut



Shadow volumes

Fördelar:

• Inga upplösningsproblem, bra “hårda skuggor”

• Bygger helt på scencilbuffern

• Klarar självskuggning

• Klarar alla riktningar

Nackdelar:

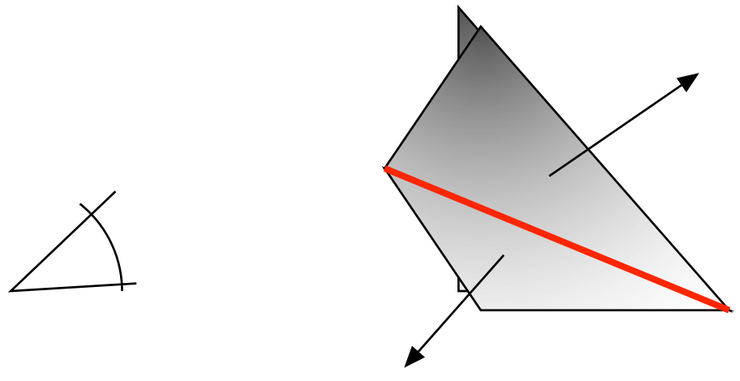
• Bökigt att beräkna volymen

• Många renderingspass över samma yta

Shadow volumes med hjälp av granndata

Half-edge eller liknande gör det lättare att hitta skuggvolymen! Sök grannar, detektera kant!

Ett pass över geometrin! Underlättar effektiv generering av skuggvolymer!



Kant = två grannpolygoner, en har framsida mot kameran, en baksida

Men det är ännu bättre än så...

Shadow volumes helt i GPU

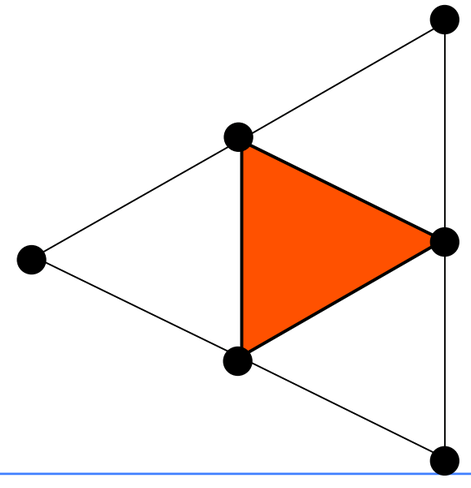
Kan utföras med hjälp av *geometry shaders*.

Granndata kan genera "adjacency"-information för geometry shaders

Adjacency kan med fördel beräknas med hjälp av half-edge.

Genererar skuggvolymen i realtid på GPU!

Nackdelen "bökigt att beräkna volymen" går bort!



Triangel med adjacency:
Tre grannvertexar följer med som del av primitivet.

Skuggkoncepten

Om vi ser det från skuggkoncepten:

- Receiver: Automatisk! Vi behöver inte identifiera receiver som vi gör med plana skuggor.

- Occluder: Manuell! Occludern måste identifieras och omvandlas till en skuggvolym.

- Mycket bra skarpa skuggor, umbra modelleras exakt. (Vi återkommer till mjuka skuggor.)

Sammanfattning

- Projektionsskuggor lätta att göra, men fungerar bara på plana ytor
- Shadow maps fungerar mycket bra, klarar de flesta scener fint, om man bygger en bit vidare från grundmetoden
- Shadow volumes ger fina skuggor, men är scenberoende, kräver specifik processning av skuggande objekt
- Bortom detta finns mer avancerade metoder för ljusspridning.

Ett tämligen svårt problem som börjar få sin lösning. En självklarhet för högklassig grafik!

Mer skuggor

Vad mer kan man göra?

Shadow volume BSP trees: En metod för att hantera många skuggande objekt i samma scen

Shadow penumbras: Metoder för att få “mjuka skuggor” i realtid

Det finns mycket att göra. Ändå bygger det mesta på resultat från 70- och 80-talet.

Flera ljuskällor

Jag har bara tagit upp scener med en ljuskälla.

Grundprincipen för flera ljuskällor är att ljus är additivt! Rendera en gång per ljuskälla, samt ambient separat eller inviktat i en eller flera.

Vissa metoder kan anpassas efter flera ljuskällor i en rendering. (Vilka?)

Mjuka skuggor

Simulera skuggor från icke punktformiga ljuskällor

- Multipassrendering
- Percentage Closer Filtering
- Smoothies
- Soft shadow volumes
- Single sample soft shadows

Mjuka skuggor

=

ljuskällor med signifikant utsträckning

= modellera penumbran

Mjuka skuggor genom multipassrendering

Enkel men ineffektiv lösning

Gör flera renderingar med ljuskällan i olika positioner.
Addera.

Kräver många renderingar innan det blir snyggt.
Omöjlig balans för många scener.

Percentage Closer Filtering

Vanligaste? Den viktigaste för “allmänbildning” i ämnet.

Relativt enkel metod som beräknar skuggningsgrad ur bilden/skuggmappen.

Uniform filtrering i bildplanet -> penumbrastorleken dåligt avbildad, konstant storlek i bildplanet.

Percentage Closer Filtering

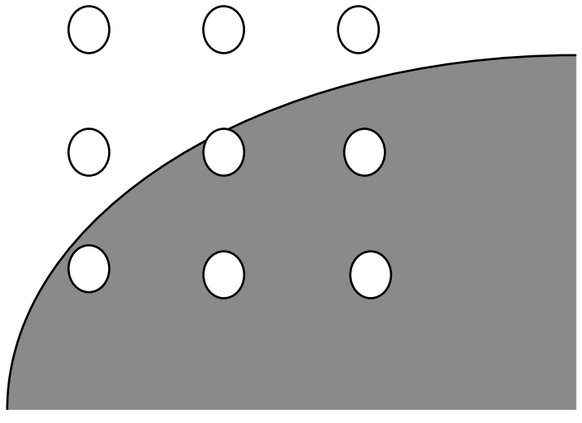
Testa n punkter kring samplingspunkten.

Om k av dessa är belysta

belysningsnivå k/n

OBS att bilddata inte filtreras! Räkning av skugga/ej skugga. Materialtextur etc är orört, får enbart ljussätttning.

Man kan INTE skapa mjuka skuggor med konventionella lågpassfilter!



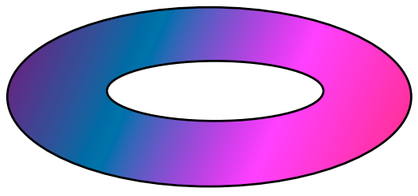
4 belysta av 9 -> ljusnivå 4/9

Smoothies

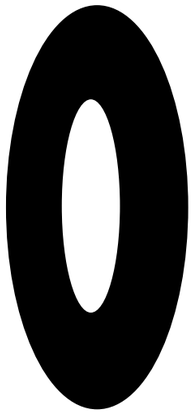
Smoothies “Fake” till och med enligt uppfinnaren!

Yttre penumbra simuleras med extrakanter i utkanten av occluderns projektion.

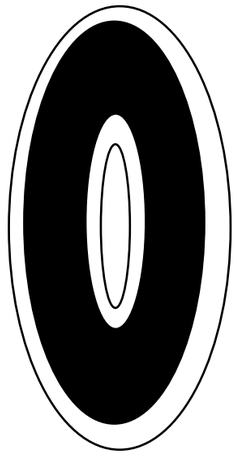
Shadow map + rendering av kanterna.



Objekt



Projicerad profil



“Smoothies” läggs till i kanterna

Soft Shadow Volumes

Soft Shadow Volumes mer ambitiös, multipassmetod med både inre och yttre penumbra. Besläktad med Smoothies.

Single Sample Soft Shadows

Besläktad med PCF. Beräknar avstånd till närmaste skugga. Vissa problem med att närmaste skugga inte alltid är den som har närmast penumbra!

En djungel av algoritmer för mjuka skuggor finns!

Slutsatser om skuggmetoderna

Mycket viktig aspekt i spel och annan modern realtidsgrafik!

Skuggvolymer ger hög kvalitet men är krävande för komplexa modeller. Scenberoende!

Skuggmappning har i sin enklaste form artefakter och riktningsberoende, men dessa kan hanteras. Ger speciellt bra resultat för mjuka skuggor.

Många algoritmer för mjuka skuggor. Skall man nöja sig med PCF eller gå längre?

Ambient occlusion

Proximity shadows

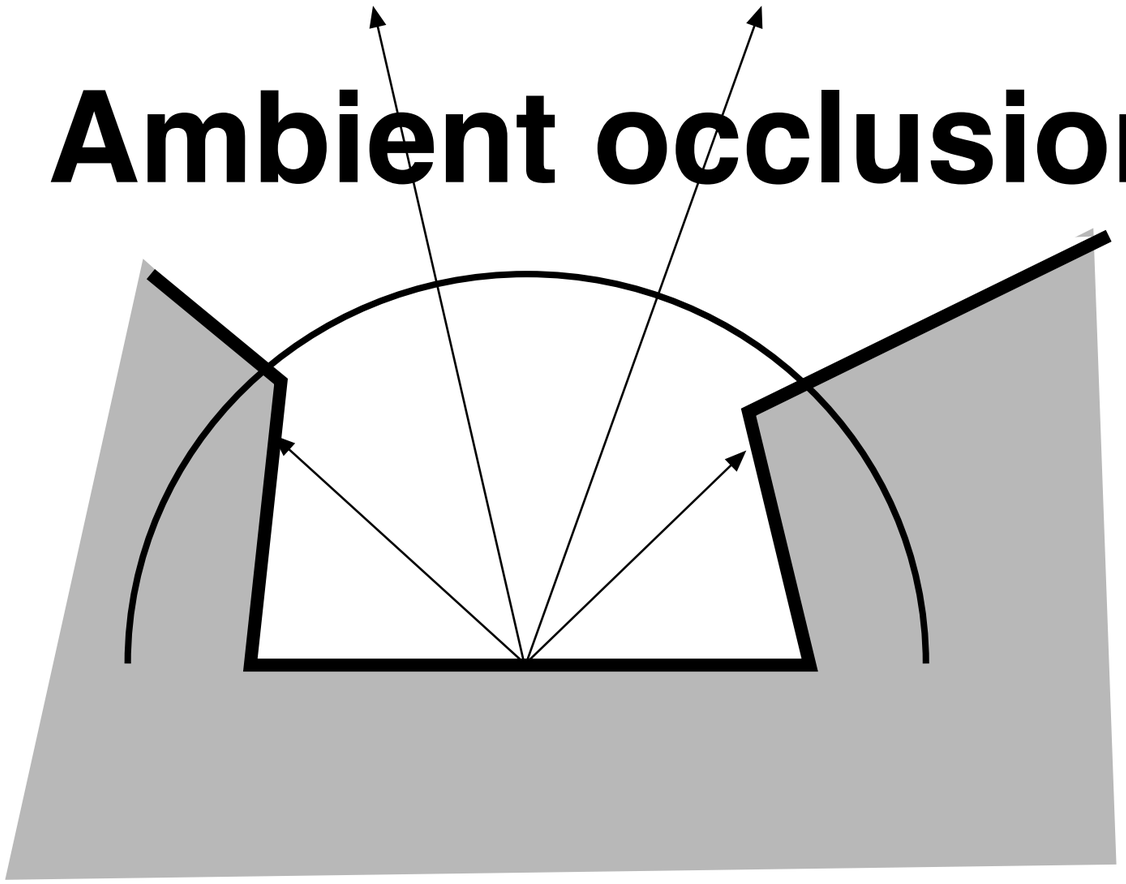
Andra ytor nära en yta - då gissar vi att den skall skuggas

Ger en bra approximation till global ljussättning.

Ambient occlusion (Proximity shadows)

Kolla om det finns
många ytor nära/
om det är ”trångt”

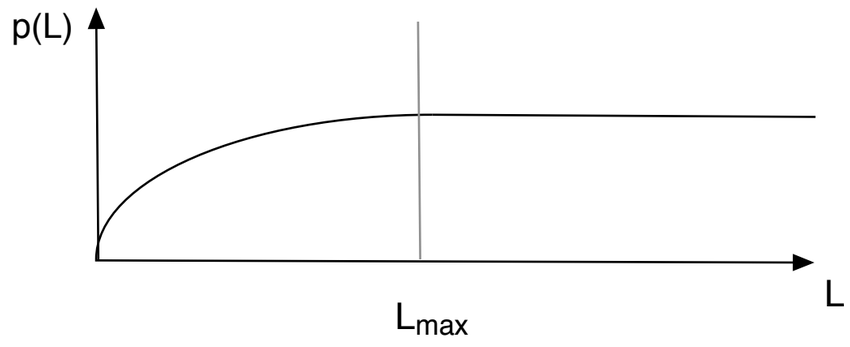
Ambient occlusion



Integrera funktion av avståndet i alla riktningar över halvklot

Avståndsberoende

Avstånd genom avklingande funktion p(L) (där L är uppmätt avstånd):



Detta betyder: Avstånd bortom L_{max} ökar inte ”obscurance”

Figur i boken från borttappad referens. Skall vara Zhukov, samma som ovanför!

Ambient occlusion i object space

Ray-casting från varje punkt

Extremt beräkningstungt, kräver en bra strukturering av världen

Förenkling av objekt effektivt och ger ytterst liten försämring

Screen Space Ambient Occlusion

Image space

Beräkna AO genom beräkningar med Z-buffer

\+ 2D-data i stället för 3D, enkel struktur

\- Skymda objekt kan inte påverka

Screen Space Ambient Occlusion

Kräver exakt match mellan Z-buffer och världen (precis som shadow buffers gör)

Scale and bias

Beräkna djup från Z-bufferinnehåll

Vertex shader, scale and bias

```
#version 150

in vec3 in_Position;
out vec2 texCoord;

void main(void)
{
  gl_Position = vec4(in_Position, 1.0);
  texCoord = vec2(in_Position / 2.0 + vec3(0.5));
}
```

Fragment shader, hämta djup

```
float readDepth( in vec2 coord )
{
  // These numbers must match the main program's
perspective projection
  float zNear = 1.0;
  float zFar = 100.0;

  float z_from_depth_texture = texture(texture0, coord).x;

// scale and bias from texture to normalized coords
  float z_sb = 2.0 * z_from_depth_texture - 1.0;

// Get back to real Z
  float z_world = 2.0 * zNear * zFar /
                  (zFar + zNear - z_sb * (zFar - zNear));
  return z_world;
}
```

readDepth() följer formlerna för normaliserade koodinater (röda boken samt PFNP)

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} \frac{2near}{right-left} & 0 & A & 0 \\ 0 & \frac{2near}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

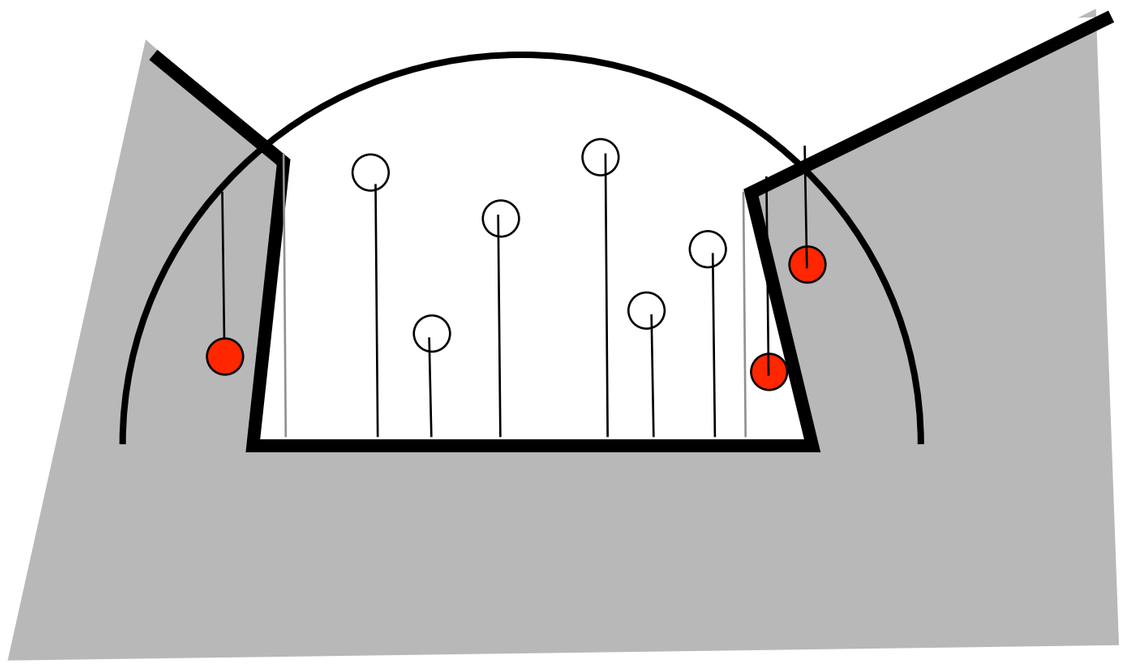
$$A = \frac{right+left}{right-left}$$

$$B = \frac{top+bottom}{top-bottom}$$

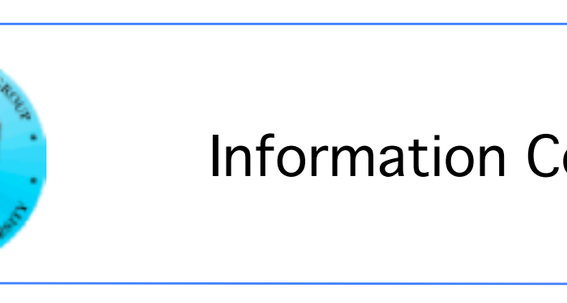
$$C = -\frac{far+near}{far-near}$$

$$D = -\frac{2 \cdot far \cdot near}{far-near}$$

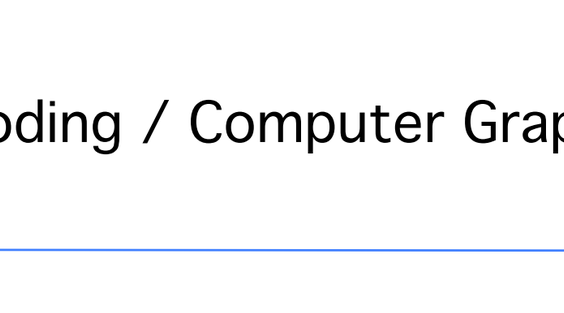
Skapa ”moln” av punkter kring centrum, mät höjd



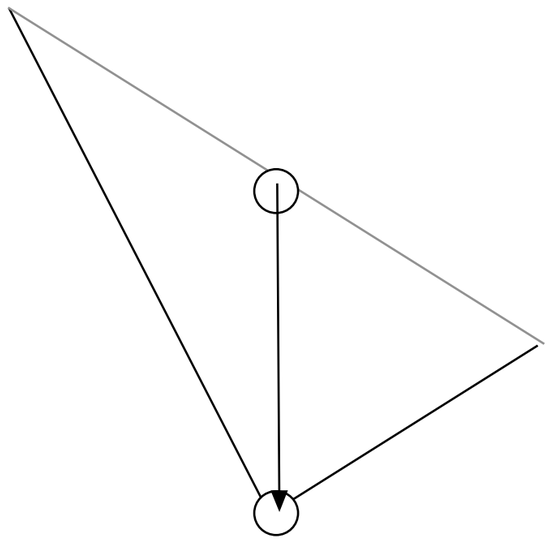
Min egen metod: Mät höjd för punkter på båda sidor, jämför medel med centrum



Mitten lägre - ingen obscurance

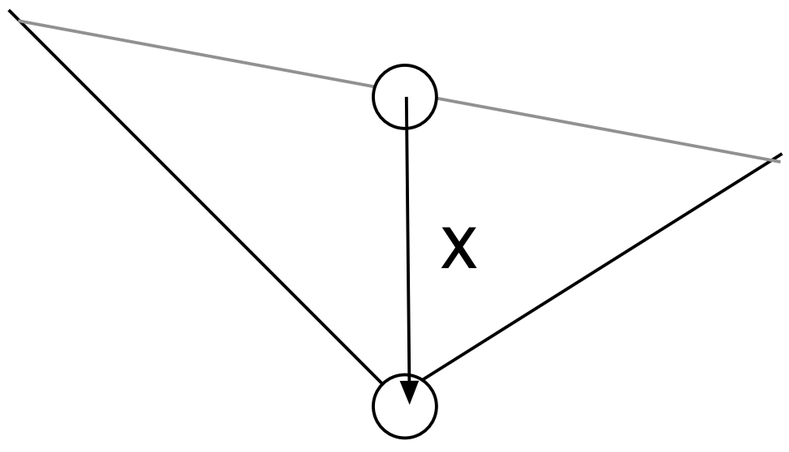


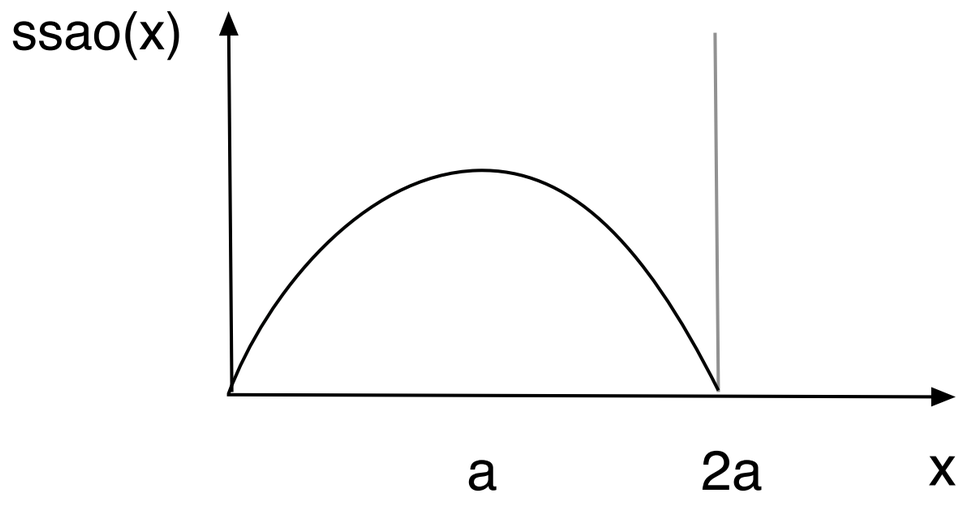
Mitten högre - obscurance



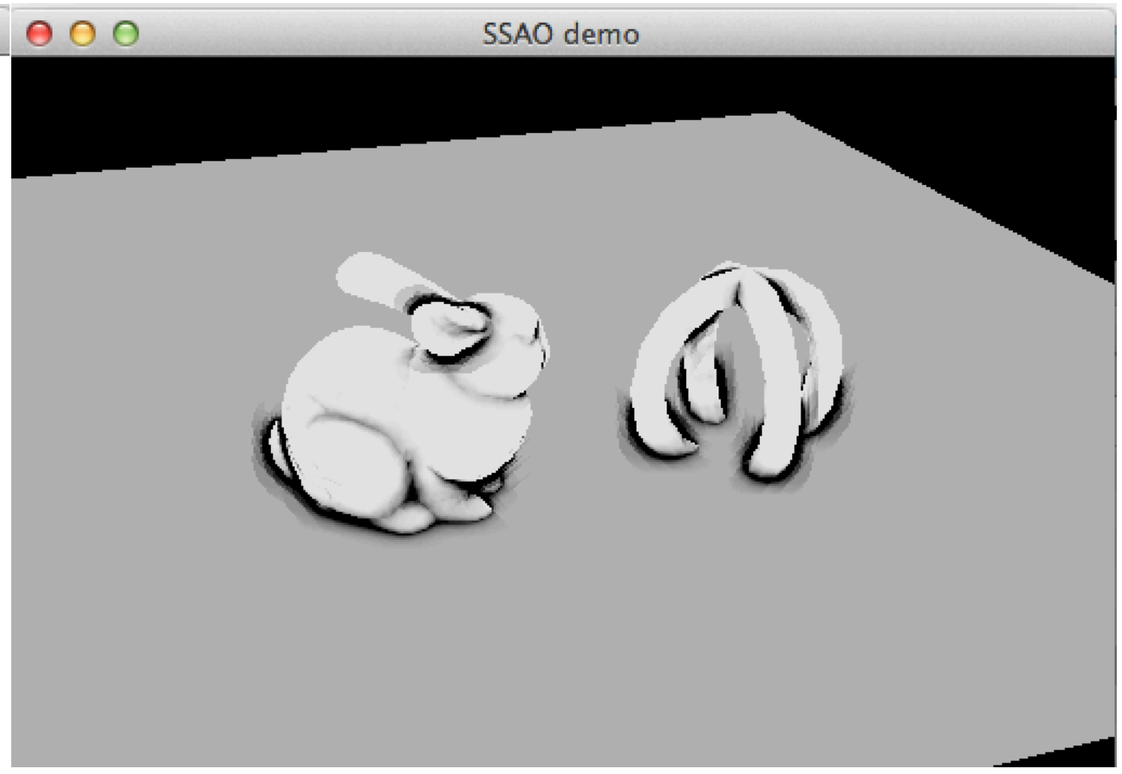
Stor skillnad - ingen obscurance (troligen skymmande objekt)

Kvadratisk funktion, ger max vid "medeldjup" skillnad





SSAO för enkel scen (överdriven)



Ambient self-occlusion

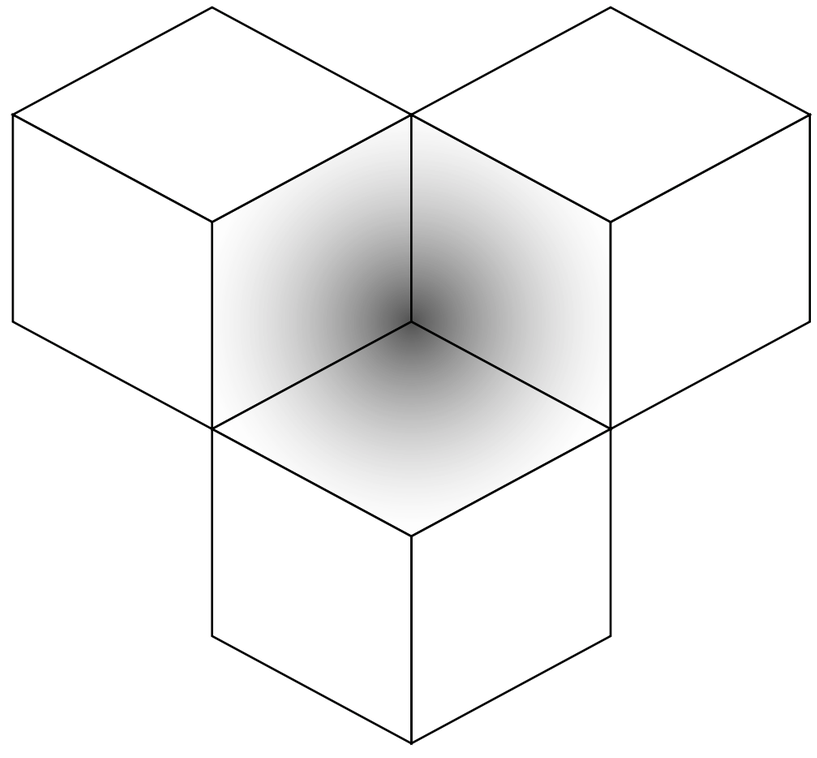
Förenklad modell

Beräkna enbart mot objektet självt

+ Kan förberäknas (men det kan radiosity också)

+ Kan beräknas mycket snabbt för enkel geometri

- Ger inga skuggor mellan objekt

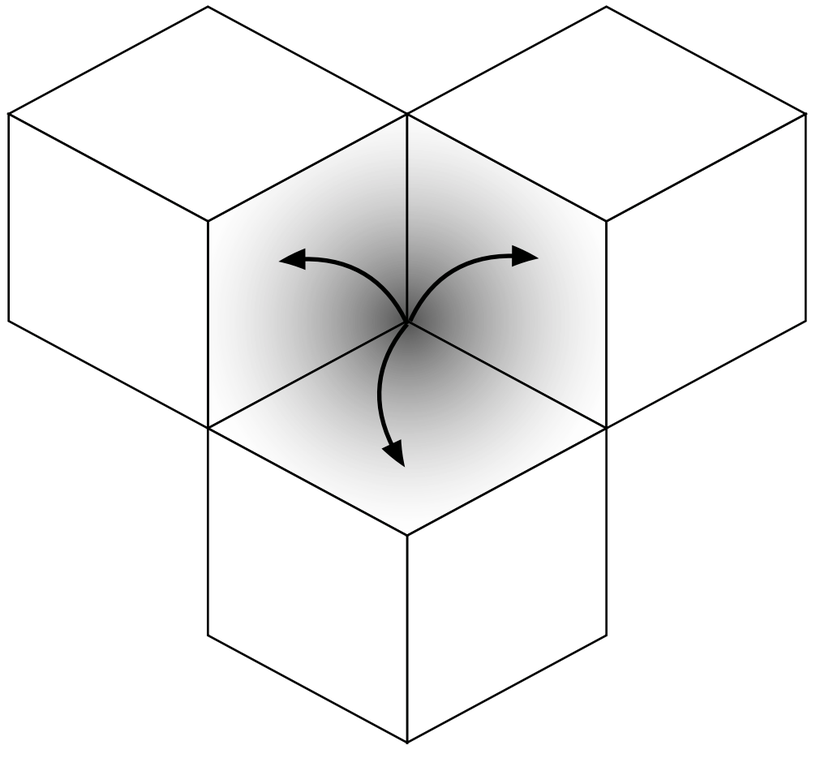


Speciellt trevlig för låg-polygon-världar!

Ambient self-occlusion ett fall för half-edge

Meshnavigering gör det snabbt och lätt att hitta grannar.

Hitta alla polygoner som gränsar till samma vertex. Undersök om de är en konkavitet. (T.ex. med höjd längs vertexens normalvektor.)



Ambient occlusion, sammanfattning

Approximation av global belysning enbart baserad på närhet till skymmande objekt

Object space, krävande men exakt

Screen Space, snabb approximation

Observera att AO alltid är en approximation!

Raytracing på GPU

Raytracing ger skuggor som del av metoden.

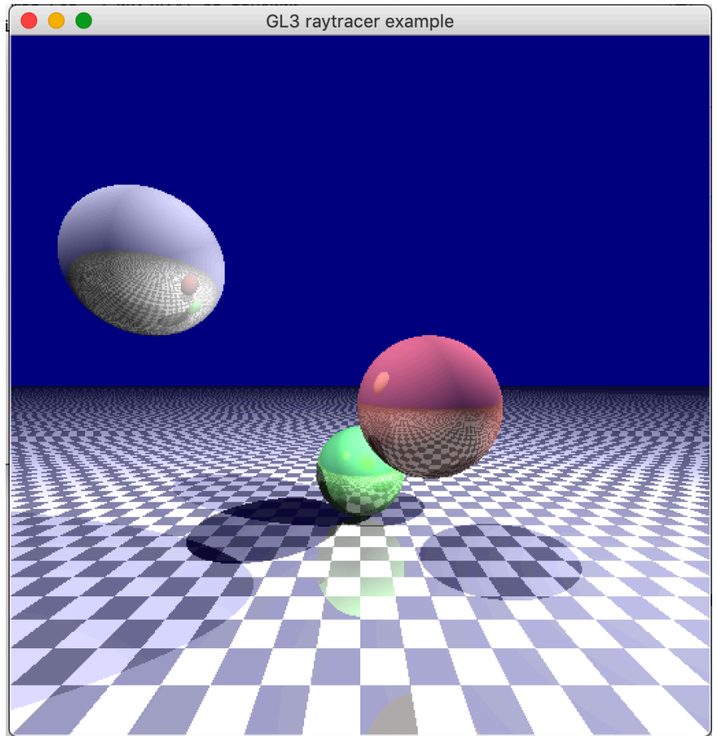
Möjligt redan på äldre kort

Aktualiserad av RTX-korten, RT cores

Accelererar BVH

Går längre än vanlig strålföljning: Photon mapping eller path tracing

Mer om detta i en senare föreläsning



Skuggenerering

Ett problem med många lösningar.

Metod kan styras av behov, prestandatillgång, typ av scen.

Mycket händer nu pga RT cores och ökade prestanda.