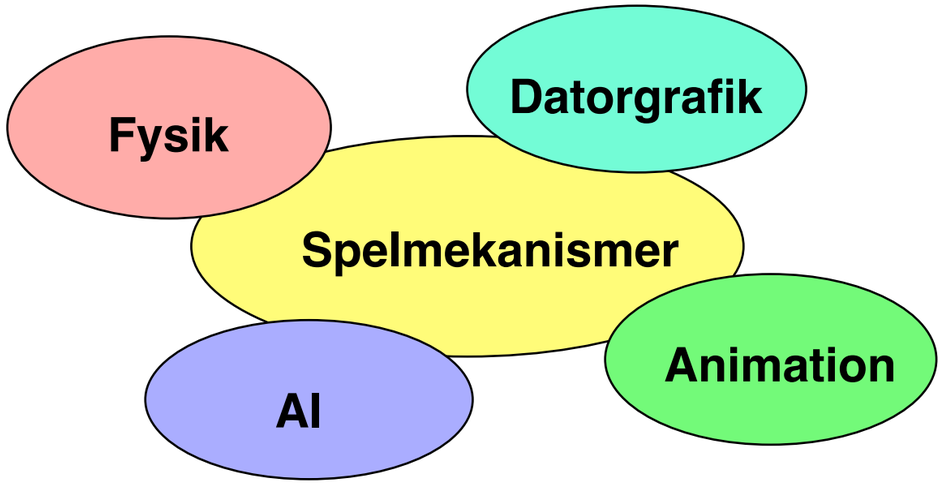
Information Coding / Computer Graphics, ISY, LiTH

TSBK 03

Teknik för avancerade datorspel

Ingemar Ragnemalm, ISY



Föreläsning 2

Buffrar, stencilbuffer
Spegel med stencilbuffer
Stencilbuffer som räknare

Texturering:
- **Andra dimensioner**
- **Multitexturering: Detail textures**
- **Projicerade texturer**
- **Rendering till textur**

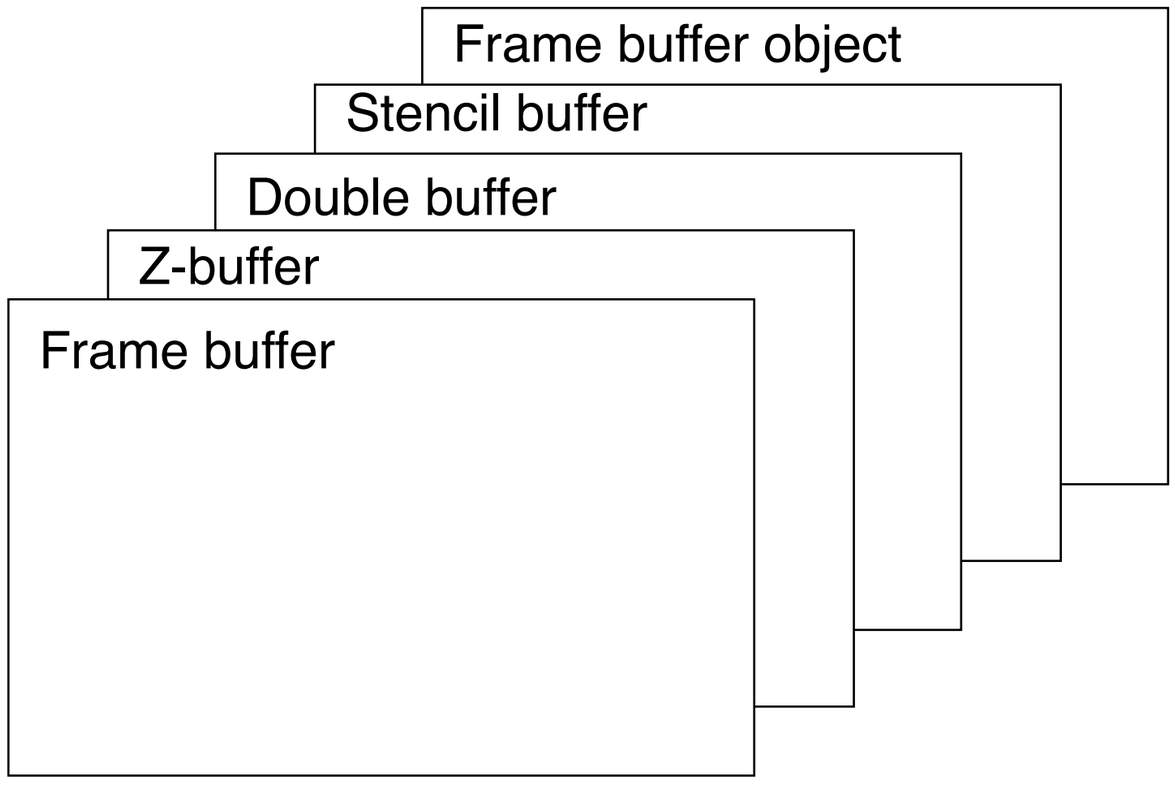
Lite annat i mån av tid.

Ytterligare grafik som följer längre fram:

- Skuggor och ambient occlusion
- Multipass-shaders, faltning
- High dynamic range, blooming
- Bättre bump mapping, parallax mapping, displacement mapping
- Stereo
- Skinning

Bildbuffrar

På GPU'n finns flera bildbuffrar, inte bara den vi ser…



De vanliga

Frame buffer: Den vanliga bildbuffern
Dubbelbuffer (back-buffer): Kopia off-screen
Z-buffer (depth buffer): Lågnivå VSD

samt icke-bildbuffrar:

Vertex buffer objects (VBO): Geometridata i VRAM
Vertex Array Objects (VAO)

Ett par till

Stencil buffer: Maskning

Frame Buffer Objects (FBO): Buffer som är textur

Shader Storage Buffer: Buffer med godtyckliga data (fö 5)

Buffer eller textur?

Texturer kan användas för mycket annat än texturdata. Är det en textur eller är det en buffer med data? Det bestämmer du!

Dock, en del specialbuffrar används i speciella steg: Stencil buffer och Z-buffer.

Stencil buffer

“Stencil”, gammal teknik för tryck. “Schablon”.

Används för att maskning, för pixelvis bestämma vilka pixlar som får skrivas.

Ritas i med vanliga ritoperationer, t.ex. rita polygoner.

Används ofta som om den var binär, men är heltal, t.ex. 8 eller 16 bitar (unsigned). Precisionen är implementationsberoende.

Många tillämpningar - vilka?

Stencil buffer

Förslag på tillämpningar:

- **Maska bort ramar, HUD…**
- **Dissolve-effekter**
- **Begränsa ritandet till ett visst objekt, viktigt för t.ex. reflektioner och skuggor**
- **CSG (Constructive Solid Geometry)**

Viktigare än man först tror?

Stencil buffer i OpenGL:

glStencilFunc(func, ref, mask);
bestämmer stencilbufferns funktion under ritande

glStencilOp(fail, zfail, zpass);
bestämmer hur stencilbuffern ändras under ritande

Tre utfall: Stencil fail, stencil pass/depth fail, stencil pass/depth pass. Olika operationer kan definieras för alla dessa fall (t.ex. inkrementera, nollställa…)

Enkelt exempel

Masking enbart

- Radera stencilbuffern
- Rita masken i stencilbuffern
- Rita andra objekt med stencilbuffer aktiv

Enkelt exempel

Måste initiera OpenGL-context med stencilbuffer:

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH | GLUT_STENCIL);
```

Radera stencilbuffern

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
```

Enkelt exempel

Rita masken i stencilbuffern

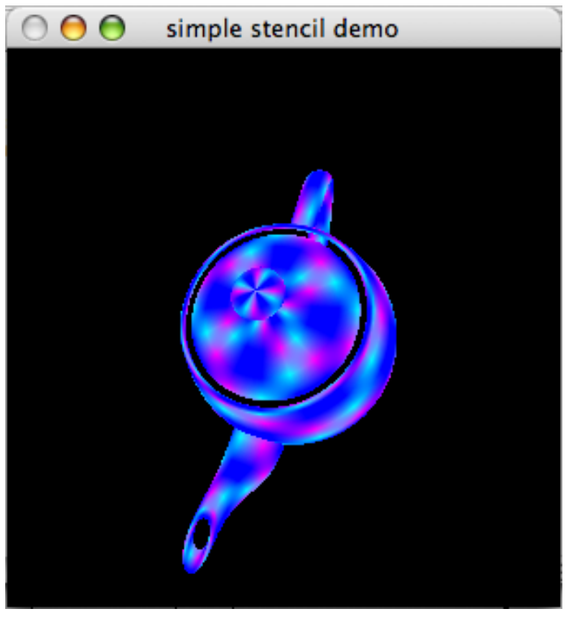
```
// Enable the Stencil Buffer
glEnable(GL_STENCIL_TEST);

// Disable Color Buffer and Depth Buffer
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDepthMask(GL_FALSE);

// Set 1 into the stencil buffer
glStencilFunc(GL_ALWAYS, 1, 0xFFFFFFFF);
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
// Draw the wired sphere in the stencil buffer only
DrawWireframeModel(sphere);
```

Enkelt exempel

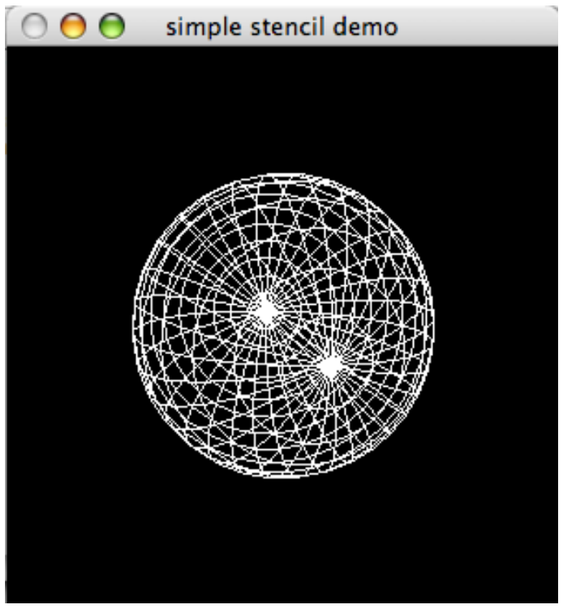
Rita andra objekt med stencilbuffer aktiv

```
// Turn on Color Buffer and Depth Buffer
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthMask(GL_TRUE);

// Only draw if Stencil Buffer equals 1
glStencilFunc(GL_EQUAL, 1, 0xFFFFFFFF);
// Keep the content of the Stencil Buffer
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

// Draw the shape
DrawTheShape();

glDisable(GL_STENCIL_TEST); // Turn off
```

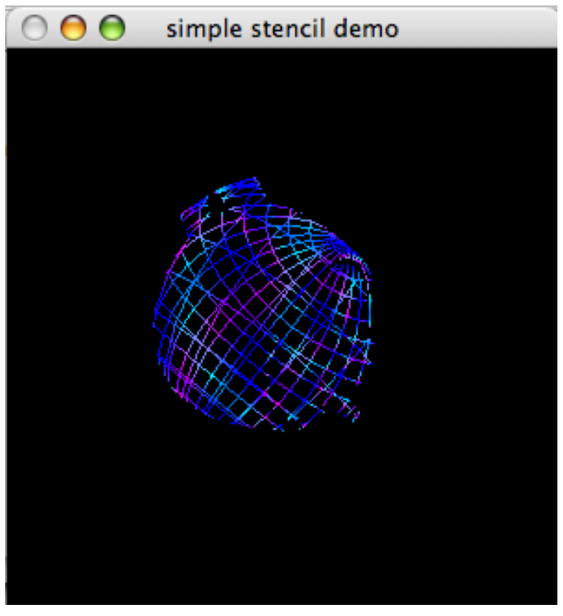
Enkelt exempel



Objekt att maska



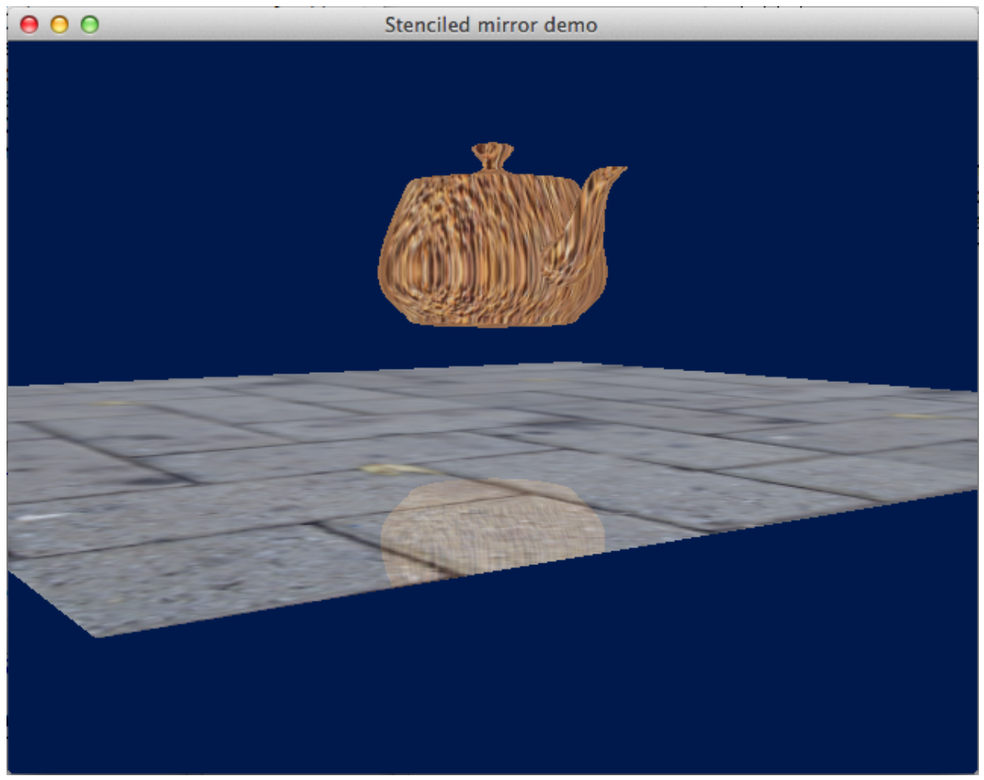
Mask



Resultat

Spegling med stencil

Exempel på vanlig användning av stencilbuffern



Spegling är mer än en speglingsmatris (S(-1, 1, 1))

- Speglade objekt utanför den speglande ytan
- Objekt bakom spegeln
- Objekt som skär spegeln
- Objekt framför som syns genom spegeln
- Ickeperfekt spegling
- Icke plana ytor (cube mapping!)

Algoritm

1) Radera stencilbuffern
2) Rita spegeln i stencilbuffern
3) Rita speglade objekt ”bakom spegeln” maskat med stencilbuffern
4) Rita spegeln (med lagom transparens)
5) Rita objekten framför spegeln

Varianter

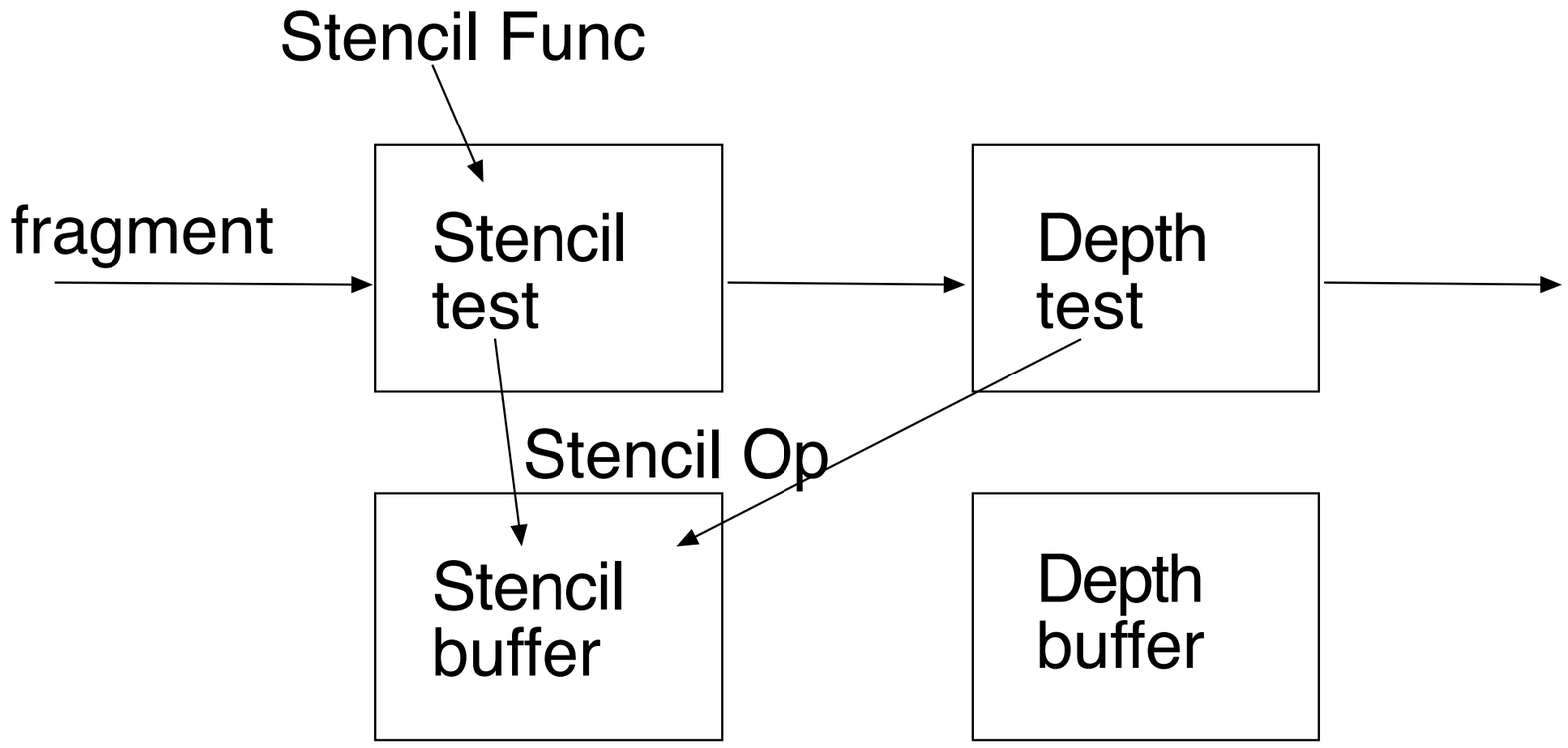
Rita spegeln först, rita speglade objekt med Z-buffer avslagen. Inga problem med objekt bakom spegeln.

Clipping av objekt som skär spegeln. (Demoner som går genom speglande portaler...?)

Besläktad metod: Maskning av plana skuggor

Kommer senare!

Stencil buffer



Sista operationerna före skrivning till frame buffer

func-parametern:

GL_ALWAYS: Default, avstängd, allt passerar.
GL_LESS, GL_LEQUAL, GL_GREATER, GL_GEQUAL, GL_EQUAL, GL_NOTEQUAL: Villkor, vad maskar eller ej?

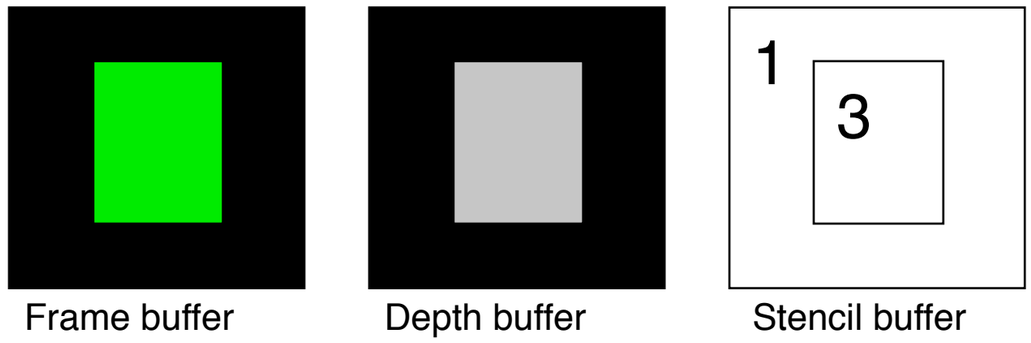
ref: Värden som jämförs med.

mask: Mask för logisk AND med ref och stencilvärde

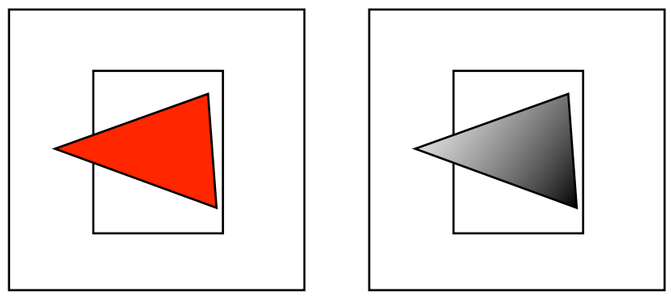
Även: glStencilFuncSeparate.

Stencilbuffer, exempel

Bild före:

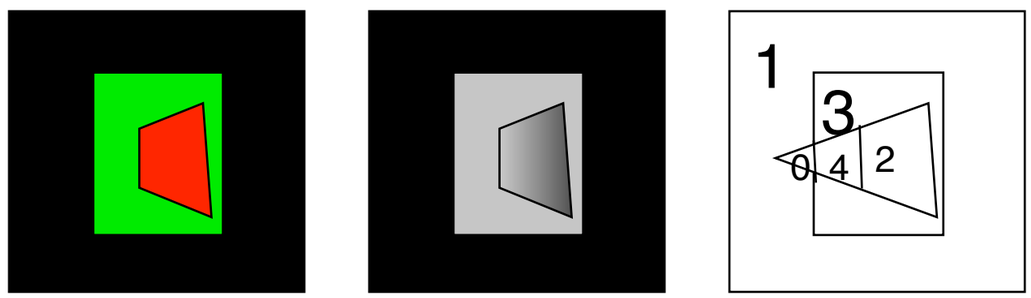


Rita triangel:



ref = 2
func = LESS
ops: fail: ZERO
zfail: INCR
zpass: REPLACE

Bild efter:

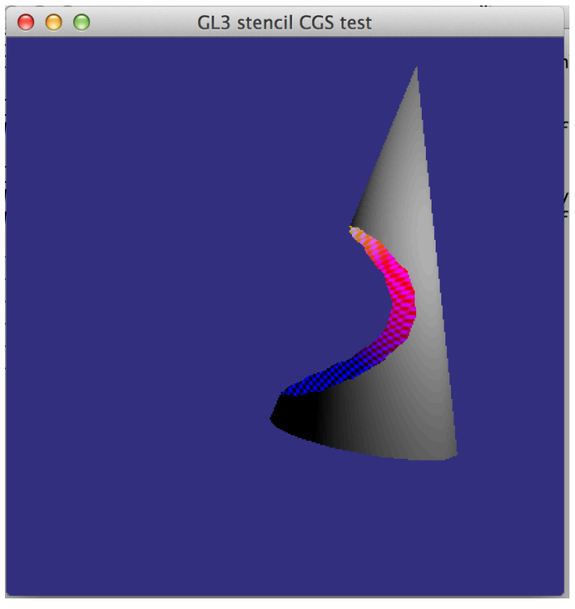


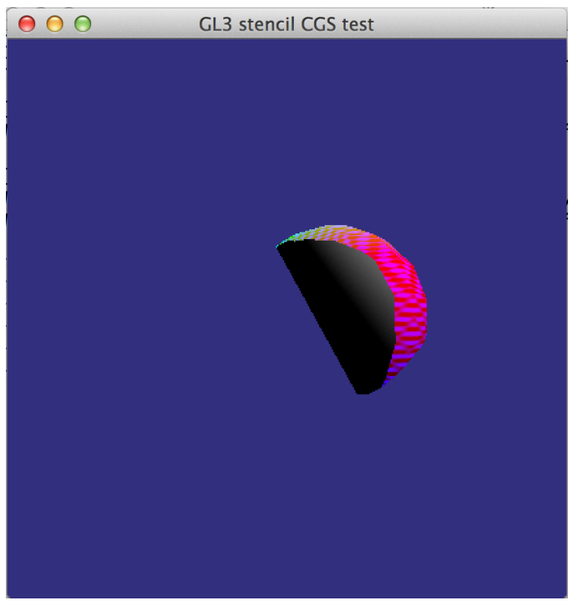
Exempel på tillämpning: Constructive Solid Geometry

Tillåter "and" och "or" samt subtrahering av ett objekt från ett annat.

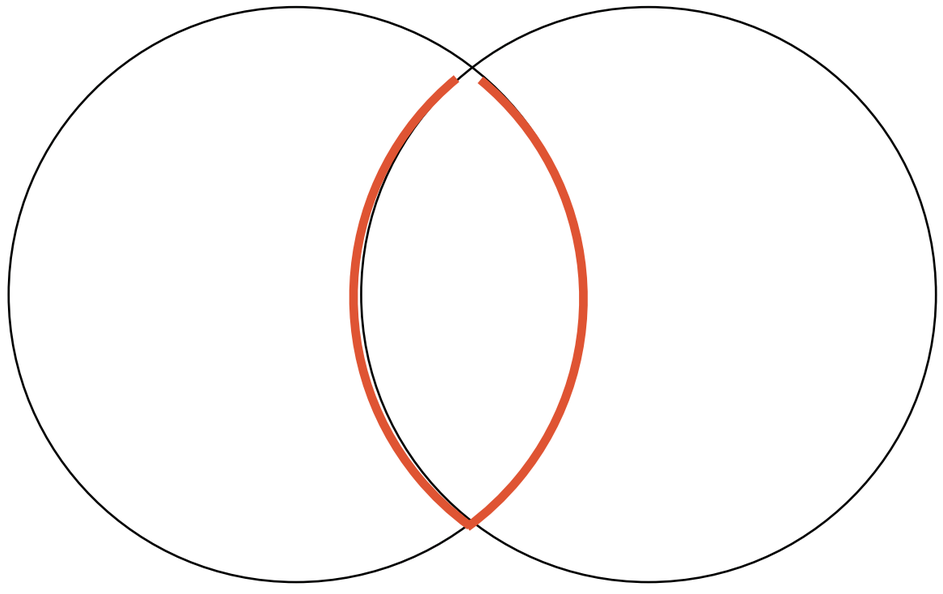
CSG med stencilbuffern

Räkna upp och ner för varje yta för att avgöra om vi är på in- eller utsidan.





Utnyttja Z-buffern och culling för att rendera rätt delar



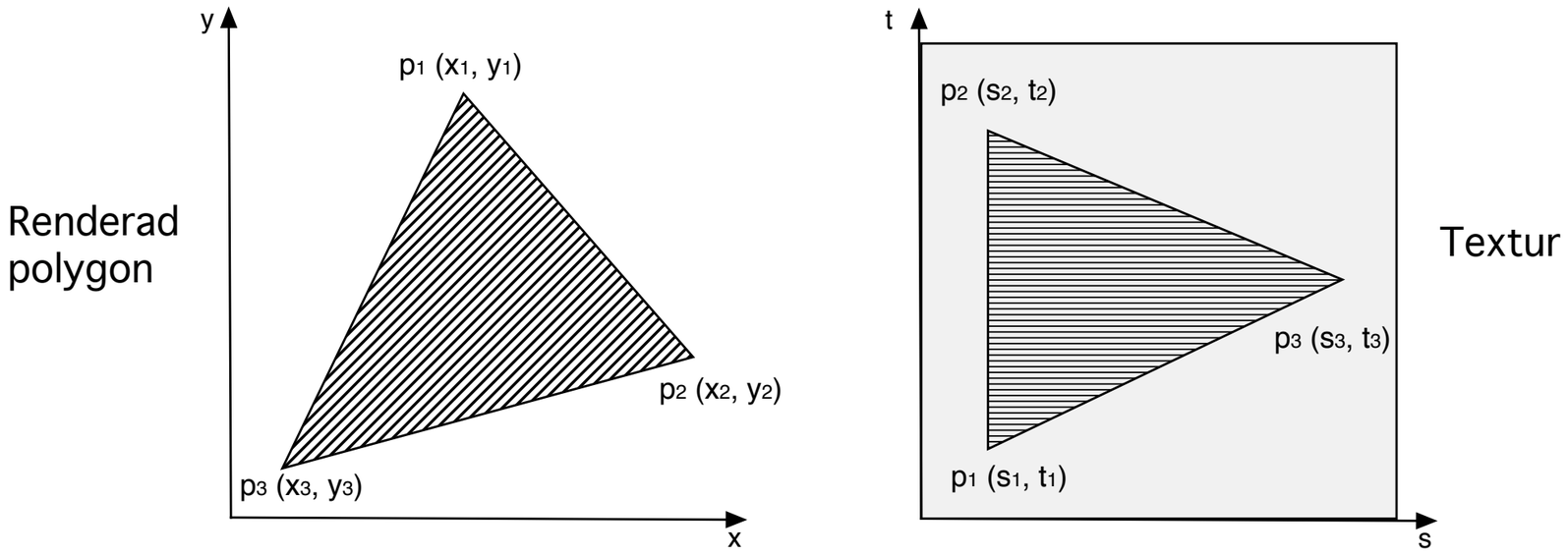
Stencilbuffern

Inga mjuka val, på eller av
Kan fungera som binär mask
Kan fungera som räknare

En av grafikhårdvarans viktigaste komponenter för ”smarta trick”.

Texturmappning

En pixelbild sträcks över en polygon och renderas med den, för detalj och realism.



- Mappningen djupberoende för rätt perspektiv
- Texturer läses in i VRAM, på GPU
- Flera texturer kan ritas på samma gång (multitexturering)
- Inte bara materialavbildningar, även ljus, reflektioner…

Vi kan väl redan:

Generera texturkoordinater:
• plan (linjär) mappning
• cylindrisk mappning
• sfärisk mappning

Texturinställningar: repeat/clamp

Texturfilter, near/far, närmaste granne, linjär interpolation…

Mipmappning

Multitexturering, splatting

Texturer i 1, 2, 3 dimensioner

1D-texturer: rad av texlar

2D-texturer: som vanligt, yta, bild

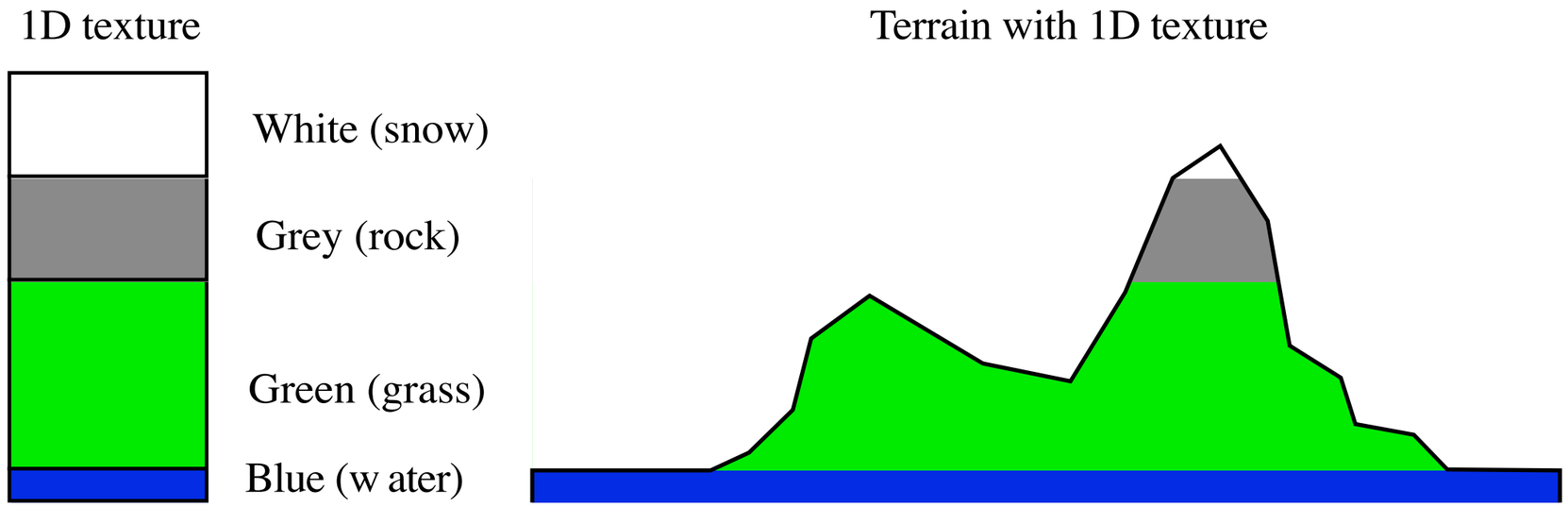
3D-texturer: volymdata

Texturer i 1 och 3 dimensioner

1D-texturer:
- Höjdfärgkodning
- Ljusdämpning

3D-texturer:
- 3D-visualisering
- Volymetriska data (rök, vatten, eld...)

1-dimensionell textur för höjdfärgkodning



Kombineras med andra texturer för detalj

Multitexturering

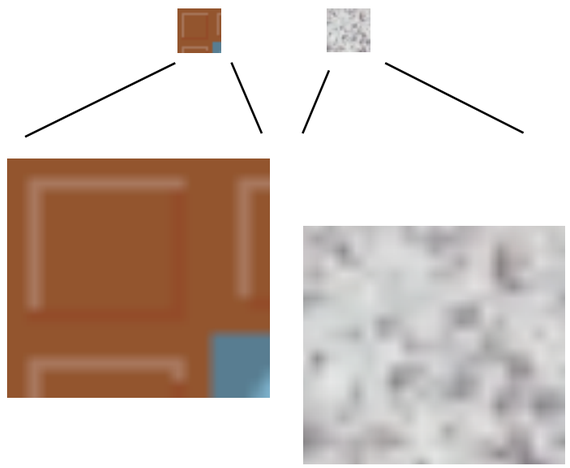
Standardmetod, lätt att utföra i fragment shader.

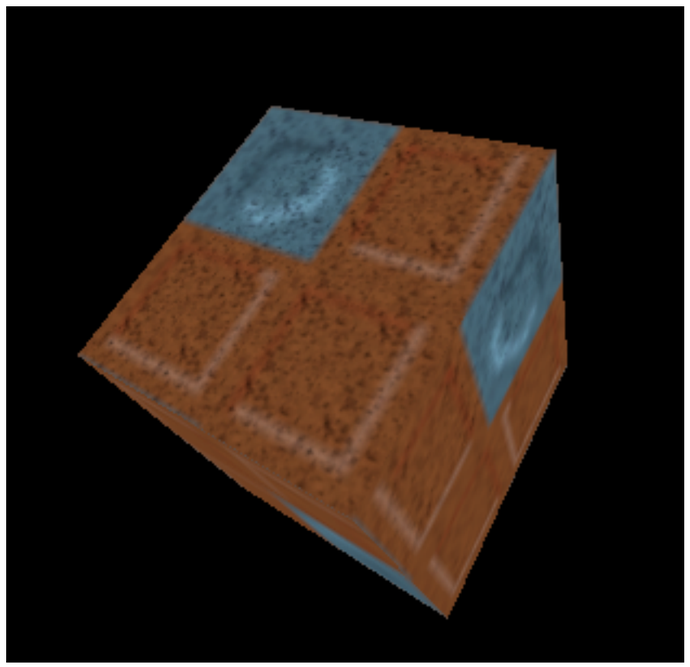
- Tona mellan olika texturer (splatting)
- Använd vissa texturer för annan information (gloss mapping, bump mapping)
- Detaljtexturer

Detaljtexturer

Kombinera högfrekvent och lågfrekvent liten textur till en stor!

Tillämpning av multitexturering.

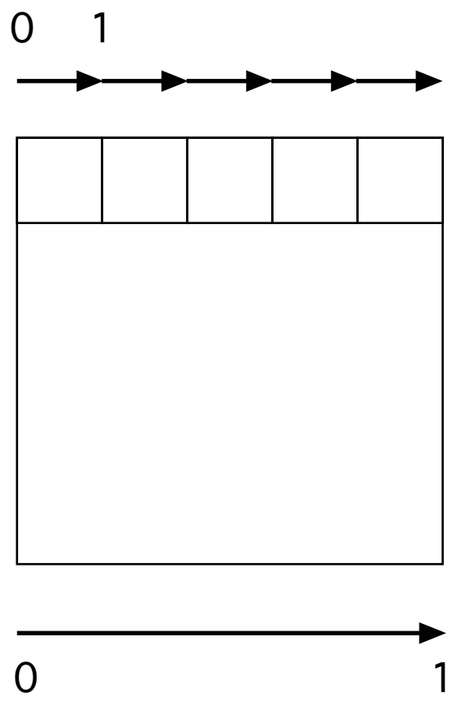




Koordinater

Multitexturering görs med olika koordinater per textur:

Lågfrekvent textur: 0 till 1
Högfrekvent textur: 0 till detailLevel
(med repeterande textur)

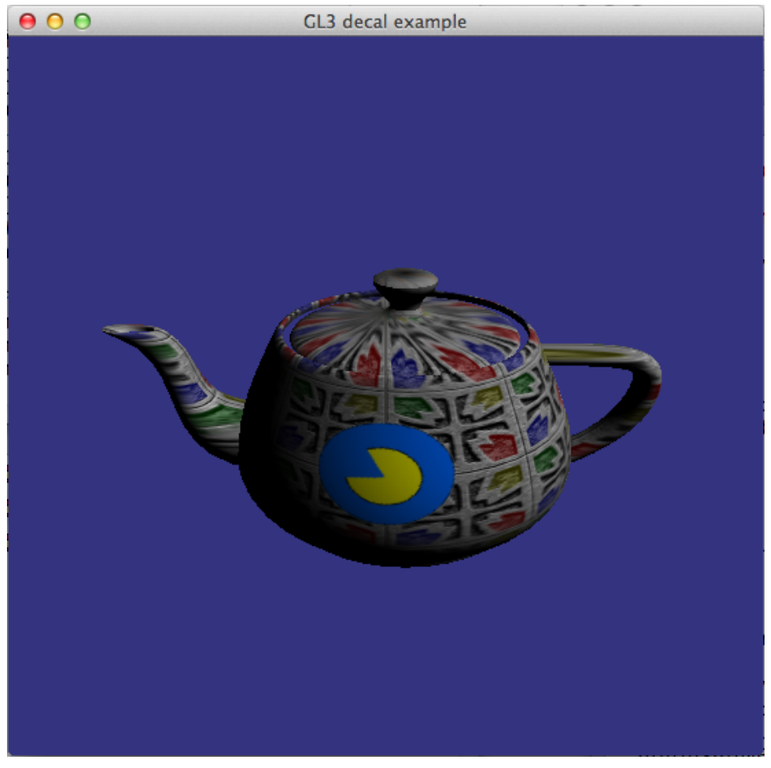


Texturkoordinater måste inte vara 1:1 med texturen!
Olika texturer kan mappas olika!

Exempel: Dekal

- Utah Teapot
- Sfärmappad yttextur
- Linjärmappad dekal med GL_CLAMP_TO_EDGE

Placera efter behov



Texturplacering

Normalt har objekt förgenererade texturkoordinater.

Man kan ganska lätt göra variationer som att skala eller translatera.

Men varför inte göra detta med matris?

Rotationsmatris för att snurra en textur!

Applicera matrisen på texturkoordinater!

Texturmatris

Enkelt exempel med texturmatris och multipasstexturering

Fast textur på hela objektet

En rörlig ”dekal”, transparent kant, GL_CLAMP för att bara ha ett varv av texturen.

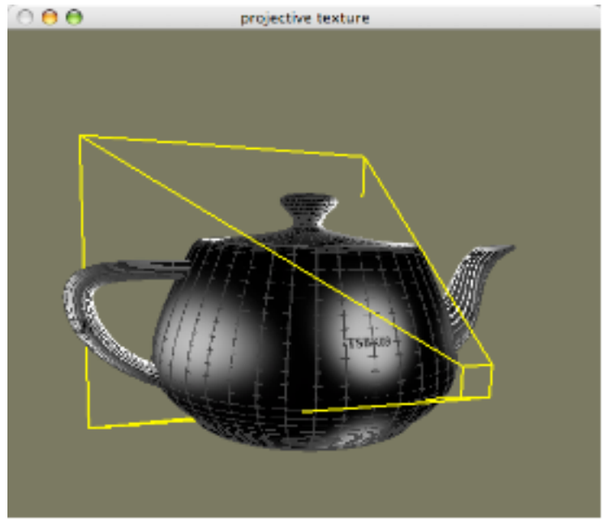
Dekalen kan flyttas och roteras med transformationer.

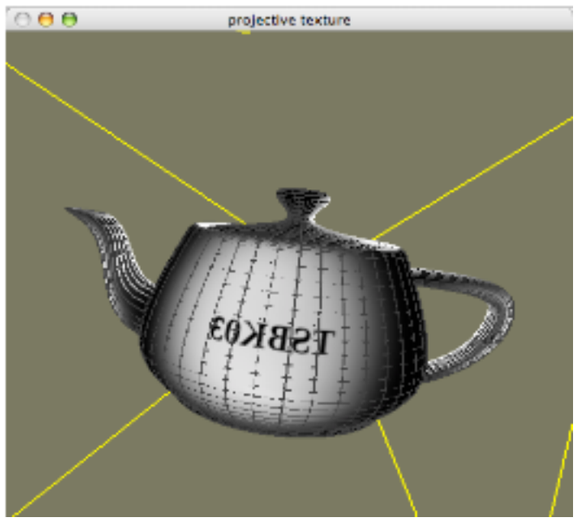
Projicerade texturer

Specialfall av texturplacering: Projiceringsmatris!

Kan appliceras både i texturmatrisen eller i objektplanen.

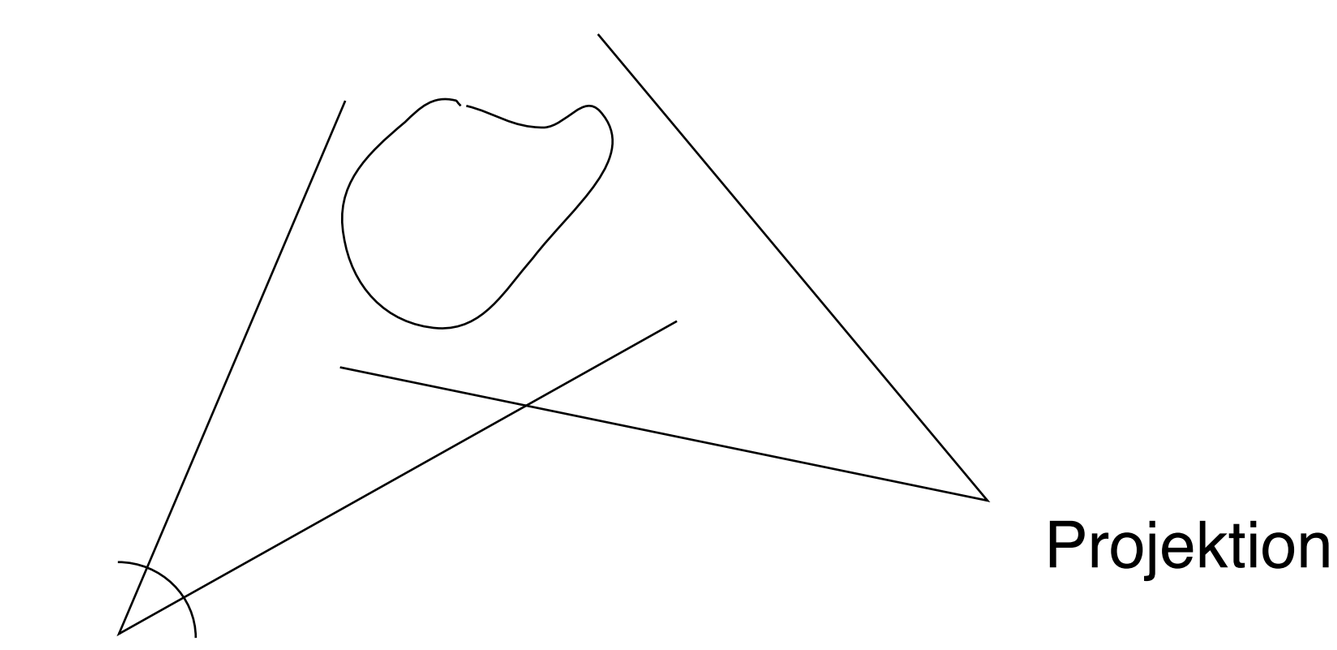
Viktiga för skuggenerering!





Projicerade texturer

En projektionsmatris skall in - men var?



Projicerade texturer

Princip:

Tag en punkt i rymden

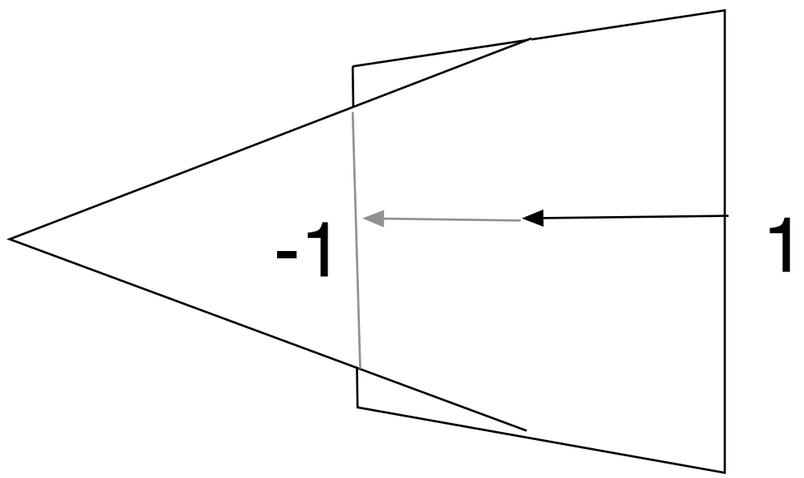
Denna punkt skall vi projicera en textur på - dvs frågan är vilken texel den motsvarar!

Kameran (projektorn) är given

Transformera punkten som till en kamera, men kameran är nu texturen!

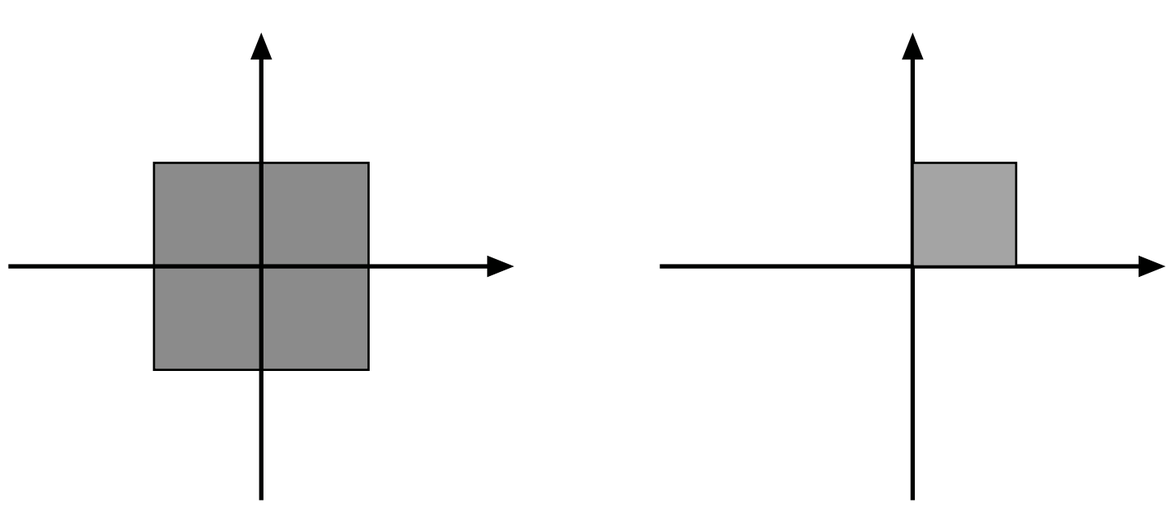
Texturkoordinaterna passar inte...

Vanlig projektionsmatris ger projektion till normerade skärmkoordinater. (-1 till 1)
Passar inte texturer! (0 till 1)



Lösning: Scale and Bias

Anpassar (-1 till 1) till (0 till 1)

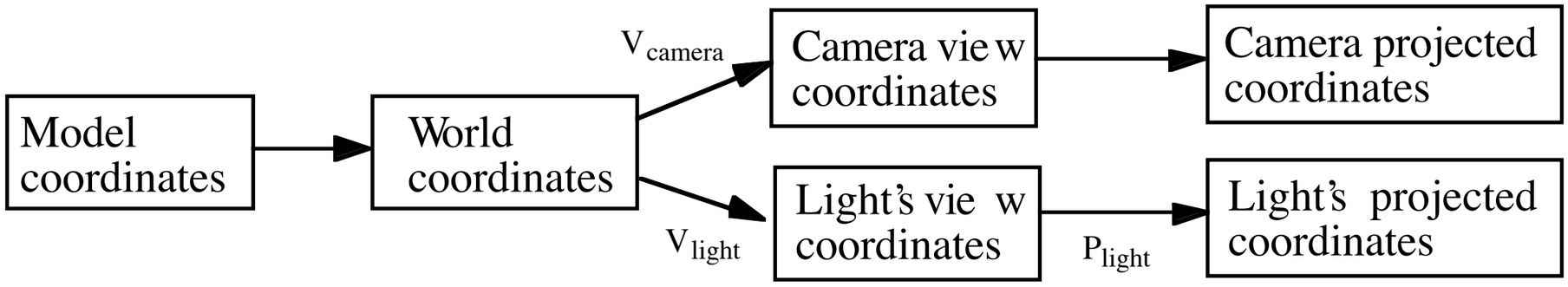


T(0.5, 0.5, 0.5);
S(0.5, 0.5, 0.5);

Projicerade texturer

Dubbla transformationskedjor, nu en till "projektorkoordinater".

Kan göras genom att backa från vykooordinater - opraktiskt.



Projicerade texturer från modellkoordinater

Bygg första steget likadant, både på modelview och texturmatrisen. Därefter unika transformationer för kameraplacering och projektion.

Projicerade texturer från modellkoordinater

Exempel

1) Sätt kameran till projektorn.
2) Multiplicera projektion*modelview, spara som texturmatrisen.
3) Sätt kameran till normal
4) Rita. Använd texturmatrisen för texturaccess.

1. Gå till projektorn

Vanlig ”look-at” men från annan punkt!

```
// Setup the modelview from the light source
modelViewMatrix = lookAtv(p_light, l_light, 0,1,0);
```

2. Skapa texturmatris

```
// Build the transformation sequence for the light source path,
// by copying from the ordinary camera matrices.
void setTextureMatrix(void)
{
 mat4 trans;
 mat4 scale;

 textureMatrix = IdentityMatrix();

// Scale and bias transform, moving from unit cube [-1,1] to [0,1]
 trans = T(0.5, 0.5, 0.0);
 scale = S(0.5, 0.5, 1.0);
 textureMatrix = Mult(trans, scale);
 textureMatrix = Mult(textureMatrix, projectionMatrix);
 textureMatrix = Mult(textureMatrix, modelViewMatrix);
}
```

3. Gå till kameran

Skapa den ”riktiga” model-to-view-matrisen!

```
    // Setup the modelview from the light source
modelViewMatrix = lookAtv(p_camera, l_camera, 0,1,0,);
```

4. Alla model-to-world-transformationer läggs på BÅDA matriserna! Rita!

```
// One torus
 trans = T(0,4,-16);
 scale = S(2.0, 2.0, 2.0);
 trans = Mult(trans, scale);
 mv2 = Mult(modelViewMatrix, trans); // Apply on both
 tx2 = Mult(textureMatrix, trans); // Apply on both
 // Upload both!
 glUniformMatrix4fv(glGetUniformLocation(projTexShaderId,
"modelViewMatrix"), 1, GL_TRUE, mv2.m);
 glUniformMatrix4fv(glGetUniformLocation(projTexShaderId,
"textureMatrix"), 1, GL_TRUE, tx2.m);
 DrawModel(torusModel, ...);
```

Vertex shader:

```
out vec4 lightSourceCoord;

in vec3 in_Position;
uniform mat4 textureMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main()
{
 lightSourceCoord = textureMatrix * vec4(in_Position, 1.0); // Transform
vertex to light source coordinates
 gl_Position = projectionMatrix*modelViewMatrix * vec4(in_Position, 1.0);
}
```

Transformerar vertexkoordinater med texturmatrisen och skickar till en varying!

Fragment shader:

```
uniform sampler2D texture;
in vec4 lightSourceCoord;
out vec4 out_Color;
uniform float shade;

void main()
{
 vec4 lightSourceCoordinateWdivide = lightSourceCoord / lightSourceCoord.w;
 lightSourceCoordinateWdivide.t = 1.0 - lightSourceCoordinateWdivide.t;
 vec4 texel = texture(texture,lightSourceCoordinateWdivide.st);

 out_Color =  texel * shade;
}
```

Utför homogena-koordinater-division. Applicerar textur som vanligt.

Sammanfattning projicerade texturer

Texturmappning genom projektionsmatris

Texturmatrisen transformerar texturkoordinater, modellerar en "projektor" för texturen

Glöm inte scale and bias!

Tillämpning: Skuggmappning! (Nästa föreläsning.)

Rendera till textur

Man kan rendera scener som inte visas direkt, utan som läggs i en textur för att användas i senare renderingssteg.

Exempel på tillämpningar:

- **Environment mapping på “riktigt”. Rendera omgivningen till textur.**
- **Spatiella filtreringar**
- **Temporalfiltrering**
- **Viktigt för skuggor**

Möjliggör multipassrenderingar.

Rendera till textur

Kan göras på flera sätt:

glReadPixels + glTexImage
Dålig, går via CPU.

glCopyTexImage
glCopyTexSubImage
Kopiering inom VRAM

pBuffers (Pixel buffers)
Frame buffer objects (FBO) - preferred!
Skriv direkt till egen buffer.

Framebuffer objects, användning

Skapa referens, samma princip som för texturer:

```
glGenFramebuffers(1, &fb);
```

Ange textur:

```
glFramebufferTexture2D(GL_FRAMEBUFFER,
GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, tex, 0);
```

Dito för renderbuffer:

```
glGenRenderbuffers(1, &rb);
glBindRenderbuffer(GL_RENDERBUFFER, rb);
glRenderbufferStorage(GL_RENDERBUFFER,
GL_DEPTH_COMPONENT24, width, height);
```

Framebuffer objects, användning

Välj aktiv FBO:

```
glBindFramebuffer(GL_FRAMEBUFFER, fb)
```

Stäng av FBO, rendera till vanliga frame buffern:

```
glBindFramebuffer(GL_FRAMEBUFFER, 0)
```

OBS! Gamla OpenGL-installationer lägger till en massa EXT-suffix!

Debugging av FBO

Under utveckling bör man fånga upp felmeddelanden från FBO. Det görs med glCheckFramebufferStatus.

Typiskt problem: Om man inte har rätt paramerar till sin textur så kan den underkännas som textur till en FBO! Destinationstexturen bör vara inställd på närmaste granne-interpolation.

Det är svårare att få ihop en fungerande FBO än man tror! (Vi har färdig kod till labbarna!)

Exempel med glCopyTexSubImage

```
void display()
{
 glBindTexture(GL_TEXTURE_2D, minitexid);

 glViewport(0, 0, width, height);

// Draw what should go in the texture
 glClearColor(1,1,0.5,0);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 DrawTextureScene();
 glFlush();

// Copy result to the texture
 glBindTexture(GL_TEXTURE_2D, tex);
 glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, width, height);

 glViewport(0, 0, lastw, lasth);

// Render final image using the generated texture
 glClearColor(0.3, 0.3, 0.7,0);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 DrawMainScene();
 glutSwapBuffers();
}
```

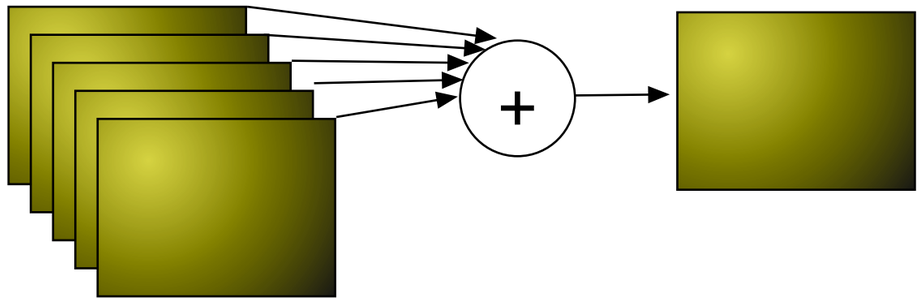
Exempel med FBO

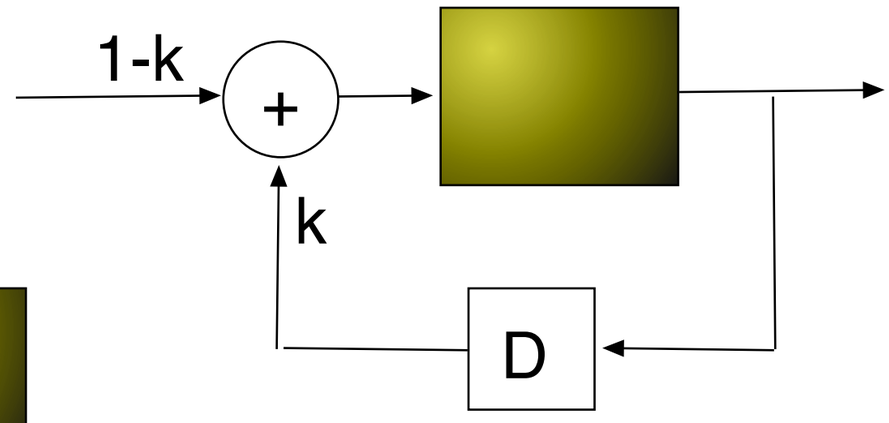
```
void display()
{
// render to the FBO
 glBindFramebuffer(GL_FRAMEBUFFER, fb);
 glBindTexture(GL_TEXTURE_2D, minitexid);

 glViewport(0, 0, width, height);

// (draw something here, rendering to texture)
 glClearColor(1,1,0.5,0);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 DrawTextureScene();

 glViewport(0, 0, lastw, lasth);

// render to the window, using the texture
 glBindFramebuffer(GL_FRAMEBUFFER, 0);
 glBindTexture(GL_TEXTURE_2D, tex);

 glClearColor(0,0,0,0);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 DrawMainScene();
 glutSwapBuffers();
}
```

Temporalfiltrering

Filtrering i tiden

Enkel tillämpning av att rendera till textur



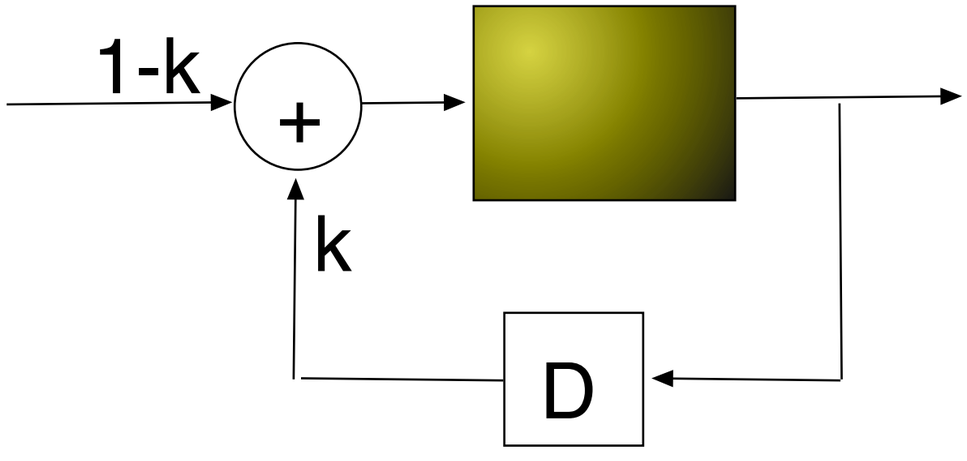


Rekursiv temporalfiltrering

Ger en ”eftersläpning”

Bilden sammanvägs med en tidigare

Kräver bara att en enda äldre bild sparas!



Rekursiv temporalfiltrering på GPU

• Rendera scenen. Godtyckling innehåll plus en bakgrundtextur, RTF-buffer. Varje pixels värde sammanvägs med bakgrundstexturen över hela bilden

Om du använder glCopyTexSubImage: Kopiera den färdiga bilden till texturen.

(Något annorlunda om man använder FBO.)

CPU-kod

Select the texture (RTF buffer)

```
glActiveTexture( GL_TEXTURE1 );
glBindTexture(GL_TEXTURE_2D, rtftex);
```

Render scene.

Copy texture:

```
glBindTexture(GL_TEXTURE_2D, rtftex);
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0,
                    0, 512, 512);
```

GPU-kod

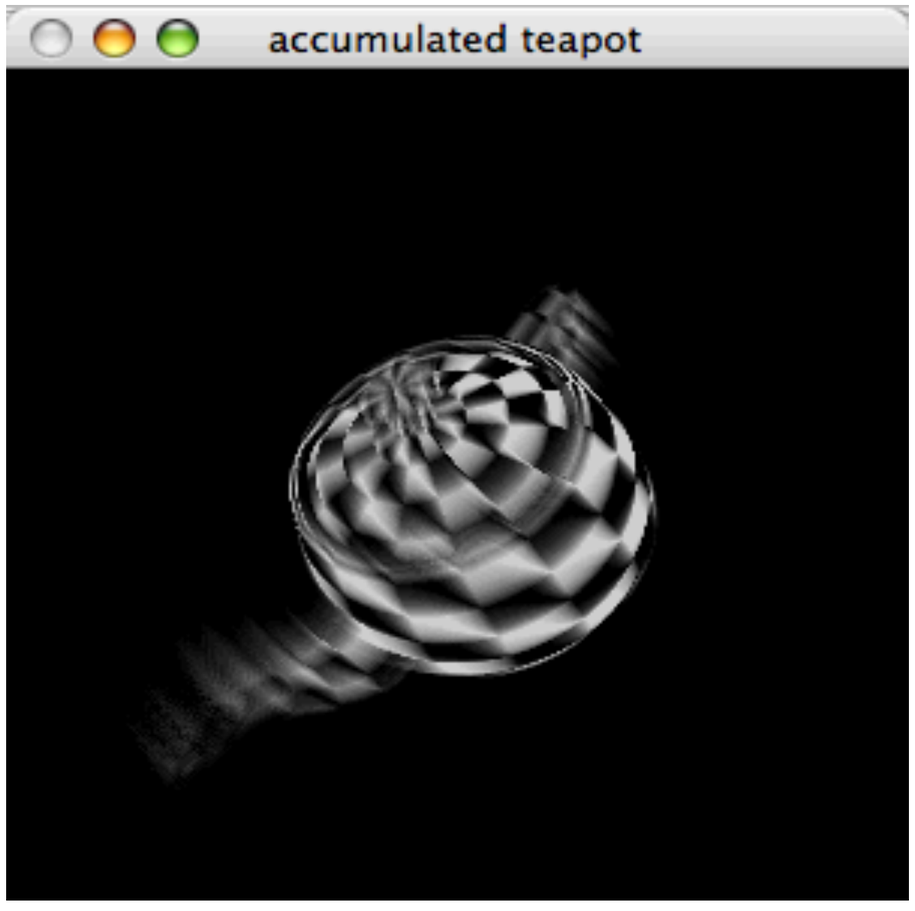
Vertexshader: Skärmkoordinater till texturkoordinater

```
out vec2 screenCoord;
screenCoord = vec2(gl_Position) / gl_Position.w / 2.0
    + vec2(0.5);
```

Fragmentshader: utför sammanvägning mellan renderingsresultat och textur:

```
outColor = texture(tex, texCoord) * (1.0-k) +
    texture(rtftex, screen- Coord) * k;
```

Rörelseoskärpa



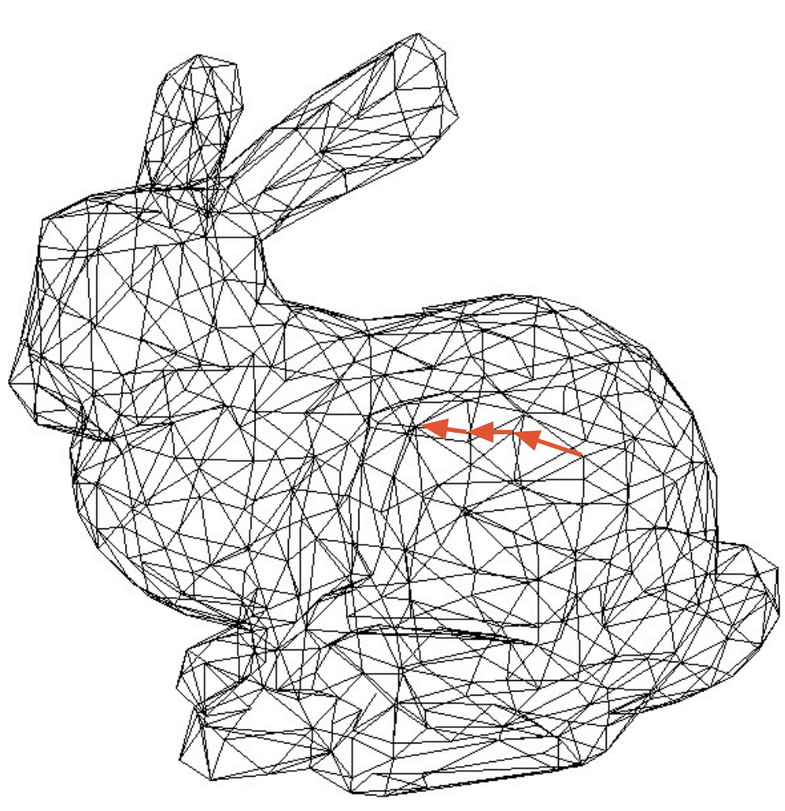
Vi har sett...

- **Stencilbuffern**
- **Projicerade texturer**
- **Rendering till textur**

=> Nu har vi viktiga verktyg för flera sorters skuggor! (Nästa föreläsning!)

Meshnavigering

Hur tar vi oss "runt" i en 3D-modell?



Enkelt problem!

Det är ju bara en fråga om att lägga till lite information om vad som är granne med vad!

Standardkoncept finns. Populär struktur: "Half-edge"

Andra finns, t.ex. "winged edge"

Half-edge

Även kallad "doubly connected edge list"

Varje "half-edge" beskriver ENA sidan av en kant, kopplat till en polygon.

Strukturen

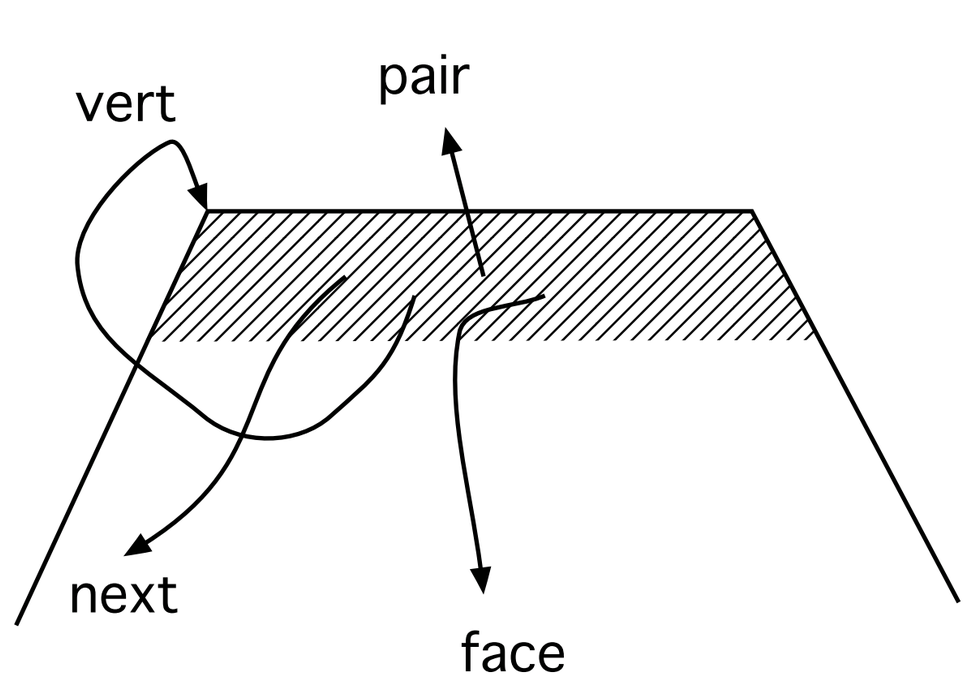
Referenser till fyra andra strukturer

- next - nästa half-edge i polygonen
- pair - andra sidan kanten
- vert - vertex vid slutet av kanten
- face - polygonen den är del av

Kort sagt: Du skall hitta vilka närliggande data du vill.

Half-edge pekar åt fyra håll

next, pair, vert, face



Gå genom alla kanter i en polygon

```
firstedge = polygon->edge;
edge = firstedge;
do
{
edge = edge->next;
}
while (edge != firstedge);
```

Gå genom alla kanter runt en vertex

```
firstedge = polygon->edge;
edge = firstedge;
do
{
edge = edge->pair->next;
}
while (edge != firstedge);
```

Tillämpningar

- **Kollisionsdetektering, e.g. support mapping**

- **LOD-beräkningar, hitta kant att ta bort, hitta sedan dess grannar.**

- **Hitta adjacency-data för geometry shaders**

- **Hitta grannar för Ambient Self-occlusion**