Uppdelningsmetoder 7(7)

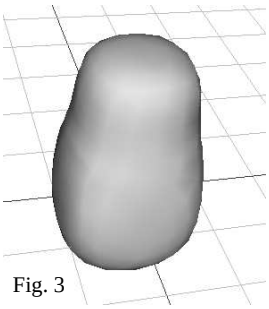


Fig. 3

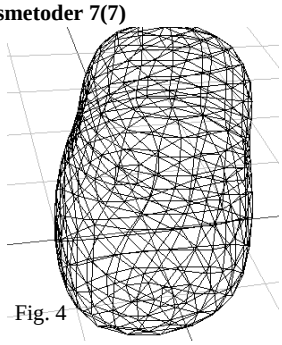
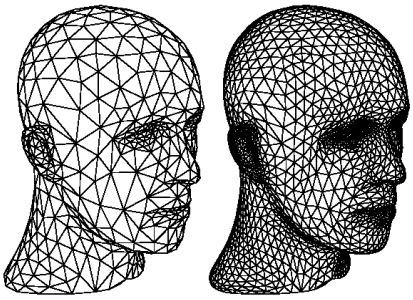


Fig. 4

Slutligen två steg i en interpolerande triangelmetod (bilden hämtad från SIGGRAPH-kursen)



P g a en miss kommer ytterligare material om kurvor och ytor med splines här.

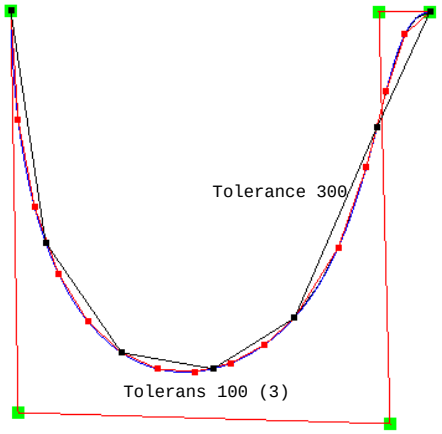
B-Splines i OpenGL: Rastreringsnoggrannhet

Denna och nästa OH (dvs OH130-131) mindre viktiga.

Rastreringsnoggrannheten (avståndet) kan styras med

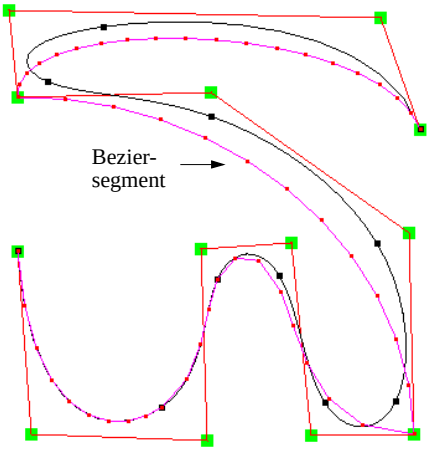
```
gluNurbsProperty(theNurb,
GLU_SAMPLING_TOLERANCE, värde);
```

Standardvärdet är 50. I figuren nedan har vi använt 100 och 300 (även 3, som i stort sett sammanfaller med 100). Värdena kan ha litet olika betydelser beroende på övriga inställda egenskaper, men normalt innebär de maximal kurvlängd räknad i bildpunkter. För 100 och 300 har vi satt ut **samplingspunkterna** (se även separat OH, ej med 2005).



B-Splines i OpenGL: Interpolation i inre punkter

Man låter tre skarvar sammanfalla (utan att ytterligare öka antalet skarvar). Har man successiva sådana trippelskarvar blir motsvarande segment en Bezier-approximation. I figuren nedan finns dels en vanlig B-splines-approximation (med sammanfallande skarvar bara i ändarna), dels en med två trippelskarvar på varandra. I det senare fallet har jag satt ut även samplingspunkterna vid kurvgenereringen.

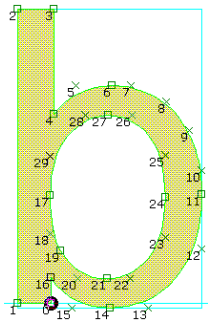


Typsnitt

Kurvapproximationer används inte bara i traditionellt ingenjörsarbete. En sentida tillämpning gäller typsnittsproduktion. Fram till 1980 fanns normalt bara ett typsnitt och i en enda storlek för dator-skärmar och skrivare och det låg i rasterform i ett läsminne (ROM). Mac hade när den kom 1984 åtskilliga typsnitt och i olika storlekare. Dessa var tillgängliga i rasterform. När man skulle skriva i en storlek som inte fanns färdig, gjordes omskalningar utifrån ett eller ett par befintliga storlekar. Resultatet blev i sådana fall halvdant. I laserskrivare med PostScript används i stället s k konturtypsnitt (eng. outline fonts). Varje tecken i ett typsnitt beskrivs i matematisk form en gång och oberoende av storleken. När en ny storlek behövs genereras rastret för tecknen i typsnittet. I PostScript (Type 1 och efterföljare) sker beskrivningen med kubisk Bezier-approximation. Av olika skäl ville Apple senare ha ett eget system. Företaget har därför konstruerat TrueType (TT), som bygger på kvadratiska B-splines (i själva verket kvadratiska Bezier). Detta har sålts till Microsoft, IBM m fl.

Vidareutvecklingar är TrueType GX (Apple), TrueType Open (Microsoft) och OpenType (Microsoft och Adobe). Liksom på andra områden slåss det vilt om herraväldet.

En referens: `http://www.truetype.demon.co.uk/`



Figur: Definitionspunkter och motsvarande kurva för bokstaven b i ett visst typsnitt. Den slutliga rastreringen bygger på omfattande ytterligare manipuleringar.

B-Splines/NURBS-ytor

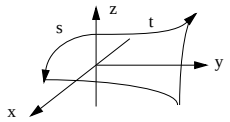
Formeln för en bi-B-Splineyta (vanligen utelämnas bi-) är :

$$\boldsymbol{P}(u,v) = \begin{cases} \sum_{0 \le i \le m} \sum_{0 \le j \le n} B_i(u) B_j(v) \boldsymbol{P}_{i,j} \\ 0 \le u \le m-2, 0 \le v \le n-2 \end{cases}$$

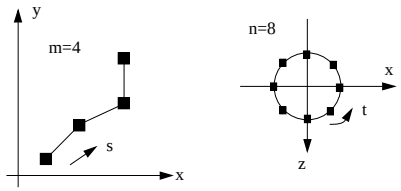
där $\mathbf{P}_{ij}$ är styrpunkterna och B_i är de vanliga basfunktionerna. Om m=n är antalet punkter lika i de båda "riktningarna" och då kan samma skarvvektor användas för båda parametrarna, annars behövs två olika. Vi är bara intresserade av det kubiska fallet.

Exempel på parametriseringar

1. Yta över en rektangel 0<x<A, 0<y<B



2. Rotationsyta



OpenGL: Ytor (kubiska B-Splines och NURBS)

OpenGL (snarare GLU) har stöd även för ytor.
OBS! Alla vektorer skall vara av float-typ (double duger inte).

- **Arbetsarea**:
```
GLUnurbsObj* theNurb;    (Java: long theNurb)
theNurb =  gluNewNurbsRenderer();
```
- **Styrpunkter**:(3->4 om NURBS)
```
GLfloat styrpunkter[m][n][3];
// Java: float styrpunkter[3*m*n];
```
- **Skarvar**:
```
int sKnotCount = m+4, tKnotCount = n+4;
GLfloat sKnots[sKnotCount] = {0,0,0,0,1,.. };
GLfloat tKnots[tKnotCount] = {0,0,0,0,1,.. };
```
- **Vikter i NURBS-fallet**:
```
GLfloat vikter[m][n];
```

Uppritning:

```
gluBeginSurface(theNurb);
    gluNurbsSurface(theNurb, sKnotCount, sKnots,
           tKnotCount, tKnots
           n*3, 3, // avstånd i GLfloats mellan
                   // styrpunkter; 4 i NURBS-fallet
           styrpunkter,
           4, 4, // gradtalet+1
           GL_MAP2_VERTEX_3 // eller motsv
    );
gluEndSurface(theNurb);
```


Ytor i OpenGL: Ett manualblad

> man gluNurbsSurface

C SPECIFICATION
void gluNurbsSurface(GLUnurbs* nurb,
GLint sKnotCount, GLfloat* sKnots,
GLint tKnotCount, GLfloat* tKnots,
GLint sStride, GLint tStride,
GLfloat* control,
GLint sOrder, GLint tOrder,
GLenum type)

PARAMETERS

nurb Specifies the NURBS object (created with gluNewNurbsRenderer).

sKnotCount Specifies the number of knots in the parametric u direction.

sKnots Specifies an array of sKnotCount nondecreasing knot values in the parametric u direction.

tKnotCount Specifies the number of knots in the parametric v direction.

tKnots Specifies an array of tKnotCount nondecreasing knot values in the parametric v direction.

sStride Specifies the offset (as a number of single-precision floating point values) between successive control points in the parametric u direction in control.

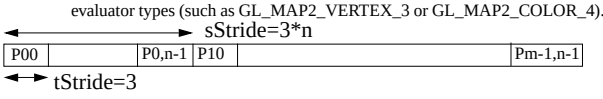
tStride Specifies the offset (in single-precision floating-point values) between successive control points in the parametric v direction in control.

control Specifies an array containing control points for the NURBS surface. The offsets between successive control points in the parametric u and v directions are given by sStride and tStride.

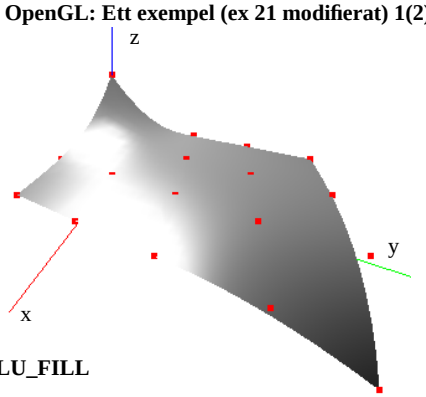
sOrder Specifies the order of the NURBS surface in the parametric u direction. The order is one more than the degree, hence a surface that is cubic in u has a u order of 4.

tOrder Specifies the order of the NURBS surface in the parametric v direction. The order is one more than the degree, hence a surface that is cubic in v has a v order of 4.

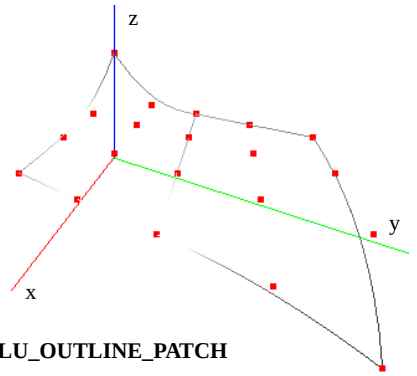
type Specifies type of the surface. type can be an of the valid two-dimensional evaluator types (such as GL_MAP2_VERTEX_3 or GL_MAP2_COLOR_4).



Ytor i OpenGL: Ett exempel (ex 21 modifierat) 1(2)



GLU_FILL



GLU_OUTLINE_PATCH

Ytor i OpenGL: Ett exempel (ex 21 modifierat) 2(2)

Styrpunkter (4 i x-led, 5 i y-led)

```
GLfloat c[4][5][3];

for (i=0; i<4; i++) {
       for(j=0; j<5; j++) {
            c[i][j][0] = i; c[i][j][1] = j;
            c[i][j][2]=2;
       }
}
c[0][0][2] = 3;  // Höj ytan vid origo
c[3][4][2] = 1.5; // Sänk diagonala hörnet
```

Skarvar

```
   GLfloat knotsu[8] = {0.0, 0.0, 0.0, 0.0,
                        1.0, 1.0, 1.0, 1.0};
   GLfloat knotsv[9] = {0.0, 0.0, 0.0, 0.0,
                       1.0, 2.0, 2.0, 2.0, 2.0};
```

Automatisk normalberäkning (fungerar för dessa ytor)

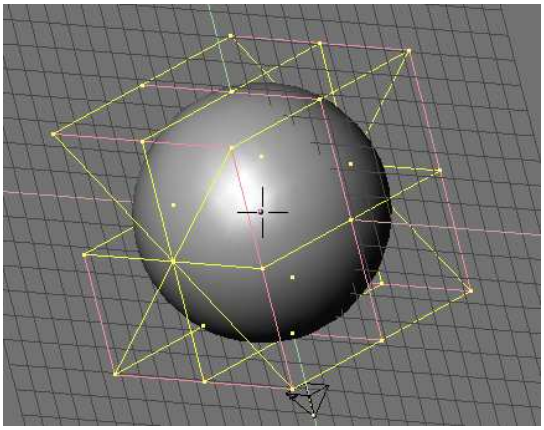
```
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);
```

Rita

```
   gluBeginSurface(theNurb);
        gluNurbsSurface(theNurb, 8, knotsu, 9,
          knotsv, 5*3, 3, &c[0][0][0], 4, 4,
          GL_MAP2_VERTEX_3);
   gluEndSurface(theNurb);
```


NURBS-ytor i Blender

Välj **Add/Surface/Sphere**. Vi har tidigare nämnt att cirklar kan repre-



senteras exakt med NURBS. Motsvarande gäller en sfär i 3D. På bilden ser vi vilka styrpunkter som behövs (som i 2D, punkterna som verkar vara extra lyser bara igenom sfären).

Man kan "redigera" sfären genom att dra iväg med en styrpunkt (tangent G och sedan musen).

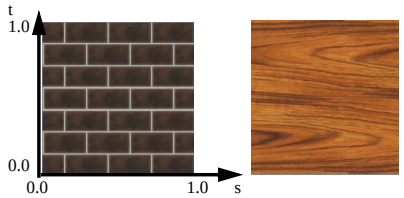
Tyvärr exporteras inga geometridata i detta fall när vi sparar (exporterar) på VRML- eller DXF-format (kollat i version 2.34).

Texturering 1(3)

Syfte: Att lägga ett mönster på en yta utan att geometriskt modellera de enskilda detaljerna.

Exempel: Gräsmatta, tegelvägg, träkloss, ...

Texturkarta:



I praktiken är texturkartan diskret, t ex 256x256 (men genereras kanske av någon procedur). De enskilda punkterna kallas **`texlar`** (eng. texel - texture element - i analogi med pixel).

Om ytan parametriserad: Säg att ytan kan beskrivas med
x = x(u,v), y=y(u,v), z=z(u,v) med 0≤u,v≤n. Gäller t ex NURBS-ytor och sfärer. Då kan vi t ex låta
T(x,y,z) = T(u/n, v/n)
eller
T(x,y,z) = T(frac(u),frac(v)),
där T med 2 parametrar beskriver texturkartan och T med 3 parametrar beskriver ytans textur, dvs vi låter i bästa datalogistil T beteckna två olika funktioner (polymorf). Detta är en rent matematisk texturering.

Texturering 2(3)

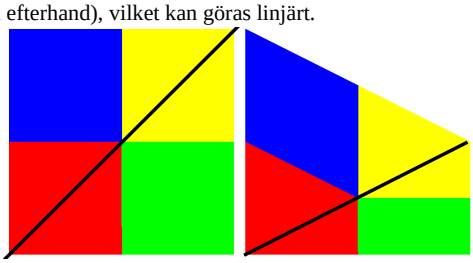
Exempel sfär med radie 1 och mittpunkt i origo: Kan beskrivas med

$$x = \cos\alpha \cdot \cos\theta$$
$$y = \sin\alpha \cdot \cos\theta \qquad 0 \le \alpha < 2\pi, -\pi/2 \le \theta < \pi/2$$
$$z = \sin\theta$$

Färgen i (x,y,z) på sfären hämtar vi från $(\alpha/2\pi,(\theta+\pi/2)/\pi)$ i texturen.

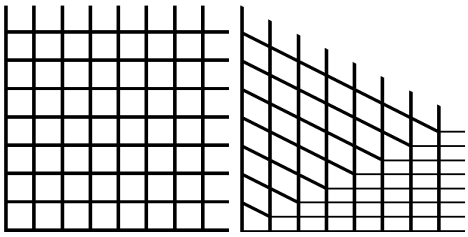
Det finns inte något entydigt sätt. I många fall vill man att påläggningen skall vara linjär (i världen), vilket ställer krav på ytan. Exempelvis kan en triangel avbildas linjärt på en annan triangel och en rektangel på en annan rektangel, men det gäller i allmänhet inte för fyrhörningar. En kvadrat kan inte avbildas linjärt på en sfär (omvändningen av kartritarnas problem), men däremot på en torus.

Exempel: Texturen till vänster avbildas på en kvadrat med hörnen (0,0), (1,0), (1, 0.5), (0,1). Man ser att det blir deformationer. Praktiskt har systemet avbildat två trianglar på två andra trianglar (de svarta dragna i efterhand), vilket kan göras linjärt.

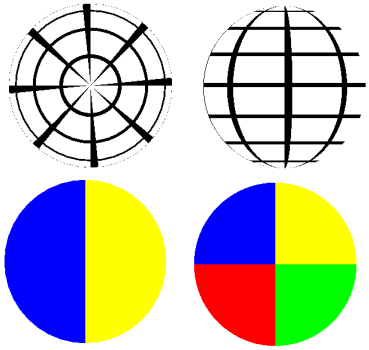


På nästa OH visas samma avbildning med en annan textur.

Texturering 3(3)



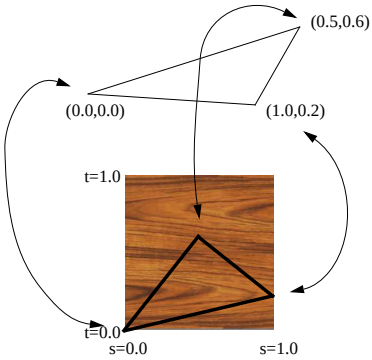
Exempel: De följande figurerna visar hur de tidigare texturerna kan se ut när de lagts på en sfär (till vänster ser vi längs z-axeln; till höger längs y-axeln; gjord med modifierad version av *glutSolidSphere* och ortoprojektion)



Texturer har visat sig vara ett kraftfullt verktyg för andra ändamål, vilket vi tar upp senare i kursen.

Texturering i grafiksystem

I grafiksystemen: Varje polygon (motsv) i världen förses med texturkoordinater i hörnen, ungefär som när vi lägger på färger eller normaler i hörnen. T ex



med innebörden att den markerade triangeln i texturkartan skall avbildas på vår triangel. Vid rastreringen (bildpunkt för bildpunkt) interpolerar (jfr nedan) man fram texturkoordinater för bildpunkten (ungefär som när man vid Gouraudtoning interpolerar fram en intensitet utifrån vad som beräknats i hörnen) och hämtar information från en (eller flera) motsvarande texel i texturkartan inför färgläggningen. Rastreringen sköts i sin helhet av grafikprocessorn och textureringen kostar inget eller obetydligt extra. Texturkartan lagras med fördel i grafikkortets texturminne.

Observera dock att interpolationen inte kan ske linjärt (om perspektiv används), se pappret "Från värld till skärm", avsnitt 10.

Texturering i OpenGL

Texturkartan: Typiskt ett rutnät med MxM punkter (kan dock vara okvadratiskt) punkter. M skulle före version 2.0 vara en 2-potens, dvs $M = 2^N$, men nu kan M vara godtyckligt. I praktiken finns någon begränsning på M, t ex att M är högst 256. Varje punkt innehåller typiskt ett RGB-värde (eller RGBA-värde; varianter finns). Som exempel en 64x64-textur

```
#define TexHeight 64
#define TexWidth 64
GLubyte Image[TexHeight][TexWidth][3];
```

I en matris av denna typ kan vi lägga tal mellan 0 och 255, varvid 255 betyder högst intensitet. Alternativt typen GLbyte med tal mellan 0 och 127.

Initieringar

Ge texturkartan innehåll: Läs från fil (konvertera vid behov) eller hårdkoda.
T ex en gul texel:

```
Image[i][j][0] = (GLubyte)255;
Image[i][j][1] = (GLubyte)255;
Image[i][j][2] = (GLubyte)0;
```

Skapa ett texturregister: Ett register för 2 texturer

```
GLuint texName[2];
glGenTextures(2, texName);
```

Vi refererar i fortsättningen till en viss textur med `texName[i]`, som i själva verket är ett heltal och kallas texturens namn. Bara en i taget är aktiv och vi gör en viss textur aktiv med

```
glBindTexture(GL_TEXTURE_2D, texName[i]);
```

Lägg in en viss textur i texturregistret: Vi skapar ett texturobjekt utan vidare beroende av den tidigare texturkartan och kopplar det till ett visst namn.

```
glBindTexture(GL_TEXTURE_2D, texName[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TexWidth,
  TexHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, Image);
//GL_RGB för RGB-värden och GL_RGBA för RGBA.
//Nollorna i glTexImage2D har ej med index 0 att göra
```


Texturering i OpenGL, forts

Sätt egenskaper: Filtreringsegenskaperna gäller per textur, men påläggningssättet gäller till dess det ändras, dvs om det skall vara olika måste det sättas vid uppritningen

```
// Filtreringsegenskaper vid förstoring respektive
// förminskning
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER
                ,GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER
                ,GL_NEAREST);
// Påläggningssätt: GL_DECAL och GL_REPLACE skriver
// över, GL_MODULATE och GL_BLEND påverkas av bl a
// framräknad belysningsfärg
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);
```

Vid uppritning

Slå på texturering:

```
glEnable(GL_TEXTURE_2D)
```

Välj textur:

```
glBindTexture(GL_TEXTURE_2D, texName[0]);
```

Ange texturkoordinat för varje hörn:

```
glTexCoord2f(0.0, 0.0); glVertex3f(0.0, 0.0, 0.0)
```

Anm 1. För vissa färdiga objekt (t ex glutSolidTeapot men inte för glutSolidCube och glutSolidSphere görs det utanför vår kontroll). Det finns också möjlighet att generera texturkoordinaterna utifrån koordinaterna i världs- eller vykoordinatsystemet.
Anm 2. Om en texturkoordinat är större än 1 upprepas (kan ändras till annat) texturen. Vi kan därigenom klä t ex en stor husvägg med en mindre textur.

Exempel: Se OpenGL-häftet, avsnitt 23. Utbyggt som GL_TEXTUR2004.c

Andra texturer: I OpenGL finns även 1D-texturer och 3D-texturer (v 1.2).