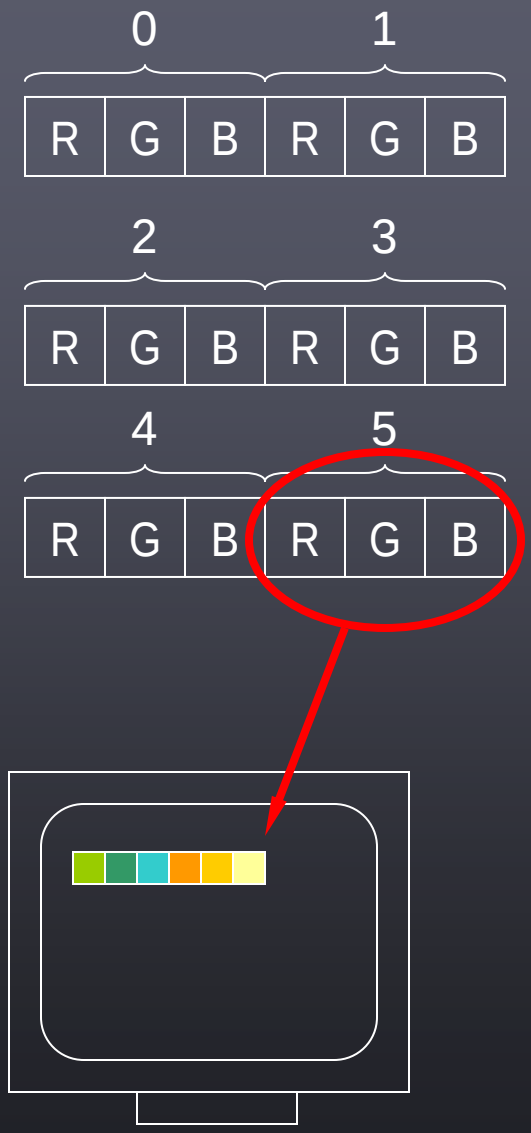
2D-grafik

Gustav Taxén
gustavt@csc.kth.se

Framebuffer

- Datorminne som lagrar information för pixlarna som ska visas på skärmen
- Grafikkortet hämtar värdena för mängden rött, grönt, och blått för varje pixel
- Styrkan hos elektronkanonen anpassas efter värdena



Pixlar

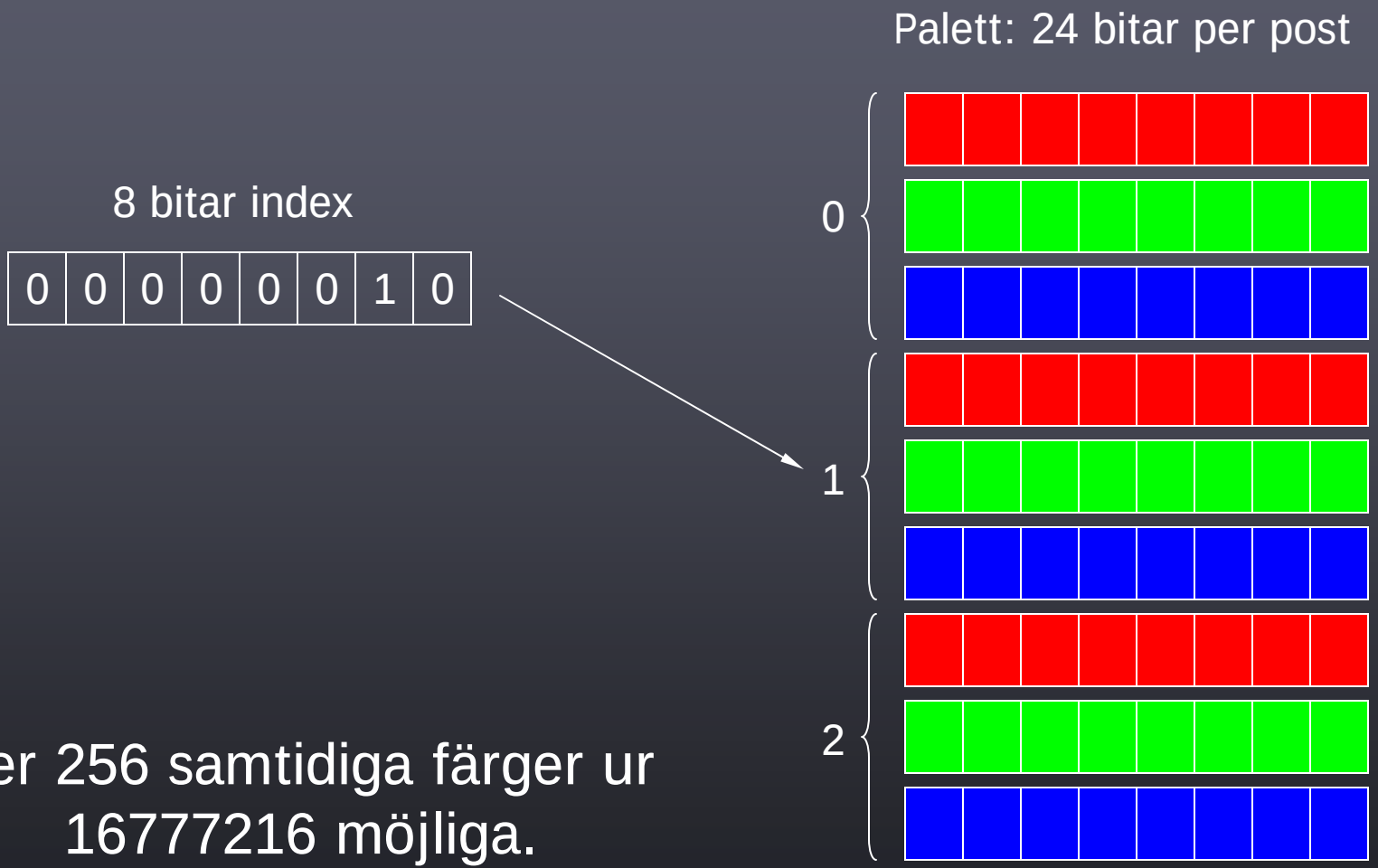
- Med **pixel** menar man oftast RGB-värdena i framebuffern
- Antalet bitar för R, G resp. B-värdena bestämmer hur många färger som kan representeras

Pixlar: 8 bitar

Man tilldelar färre bitar till den
blå kanalen eftersom människan har
svårare att se skillnader mellan blå nyanser.

Ger $2^3+2^3+2^2 = 16$ färgnyanser.
Detta är en mycket ovanlig färgkodning idag.

Pixlar: 8 bitar palett



Ger 256 samtidiga färger ur 16777216 möjliga.

Pixlar: Color cycling

http://www2.vo.lu/homepages/phahn/fractals/cycling.htm

Shifta färgerna i paletten
(oftast med hjälp av ett offsetvärde)

Klassisk 80-talseffekt i spel!

Pixlar: 16 bitar

5 bitar till R och B, 6 bitar till G eftersom människan uppfattar skillnader i gröna nyanser bäst.

Ger $2^5+2^6+2^5 = 65536$ olika färgnyanser.
Kallas ibland **hi-color**.

Pixlar: 24 bitar

1 byte (8 bitar) till R, G, och B.

Ger $2^8+2^8+2^8$ = 16777216 olika färgnyanser.
Kallas ibland **true-color**, eftersom
människan kan uppfatta ungefär så många nyanskillnader.

Pixlar: 32 bitar

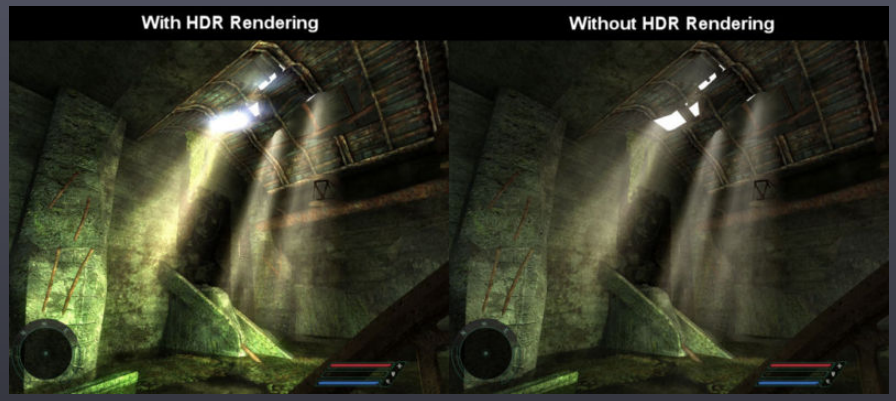
- 8 röd + 8 grön + 8 blå + 8 alpha
- 16777216 färgnyanser +
  256 genomskinlighetsnivåer

Pixlar: mer än 8 bitar per kanal

- Konverterar du en 24-bitars bild till svartvitt är du tillbaka i 256 färgnyanser!
- Manipulation av bilder kan minska antalet signifikanta siffror eller ge avrundningsfel så att du tappar information
- Idag används ofta 12 eller 16 bitar per kanal (16x4 = 64 bit)
- Ibland t.o.m. 32 bitar per kanal (128 bit)

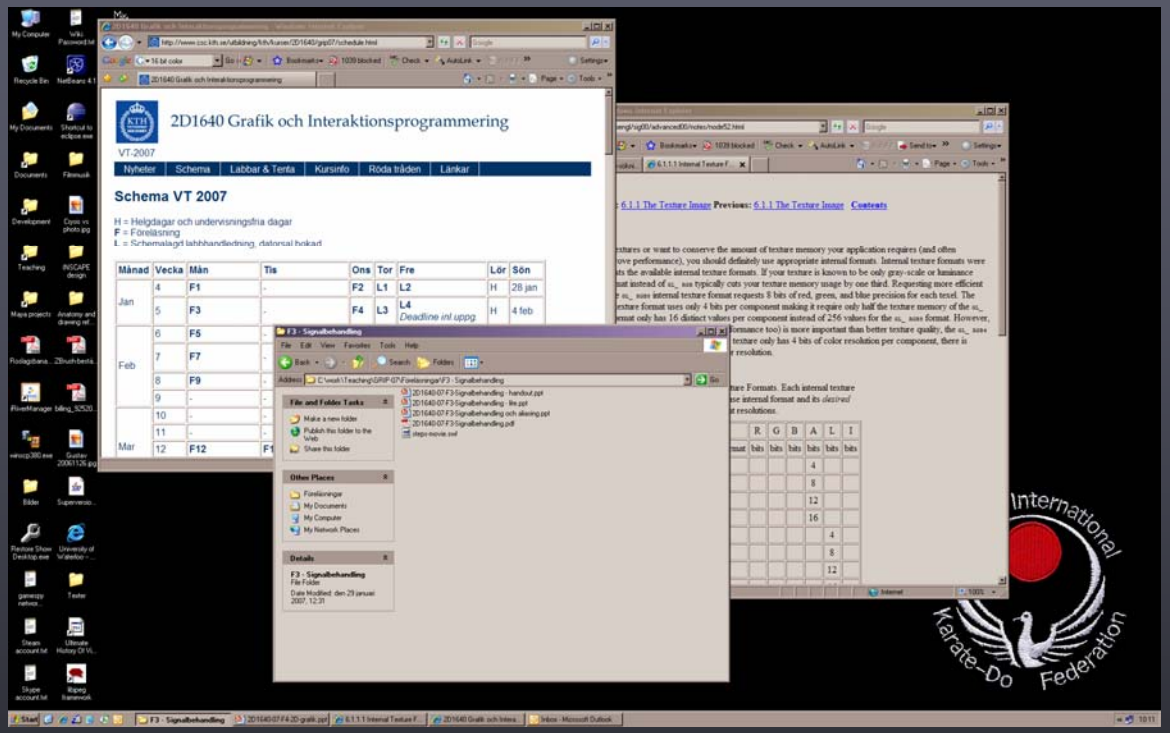
Pixlar: fixed point & floating point

- Oftast lagrar man i fixed point, d.v.s., man använder de binära värdena rakt av
- I vissa fall behöver man representera mycket stora intervall i varje kanal
- T.ex. High Dynamic Range
- Kanalerna representeras med floats, ofta 32 bitar per kanal



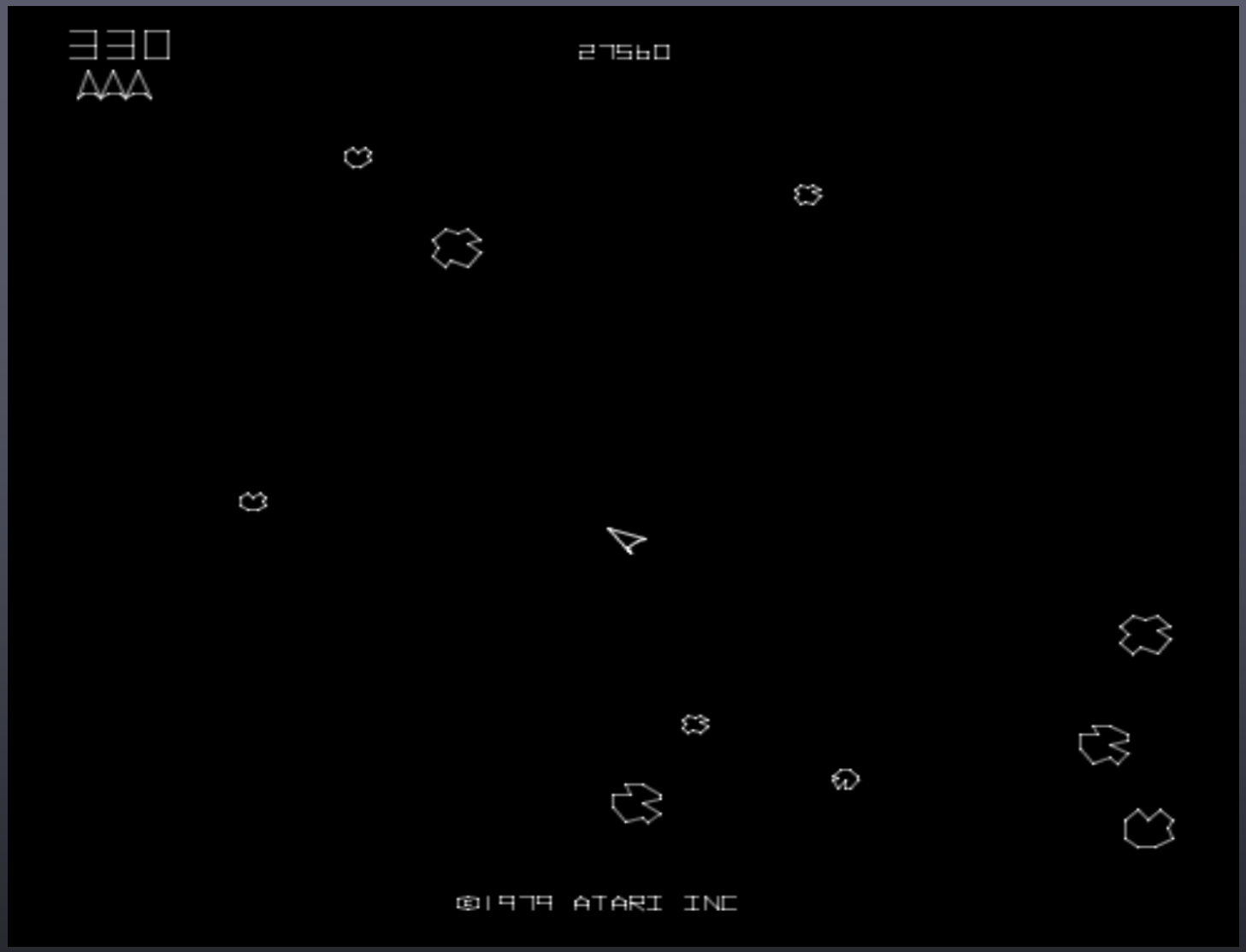
http://en.wikipedia.org/wiki/Image:Farcryhdr.jpg

Fönstersystem



För att göra det enklare för programmeraren har oftast varje fönster en egen "framebuffer".

Dessa kombineras sedan ihop i den "riktiga" framebuffern.

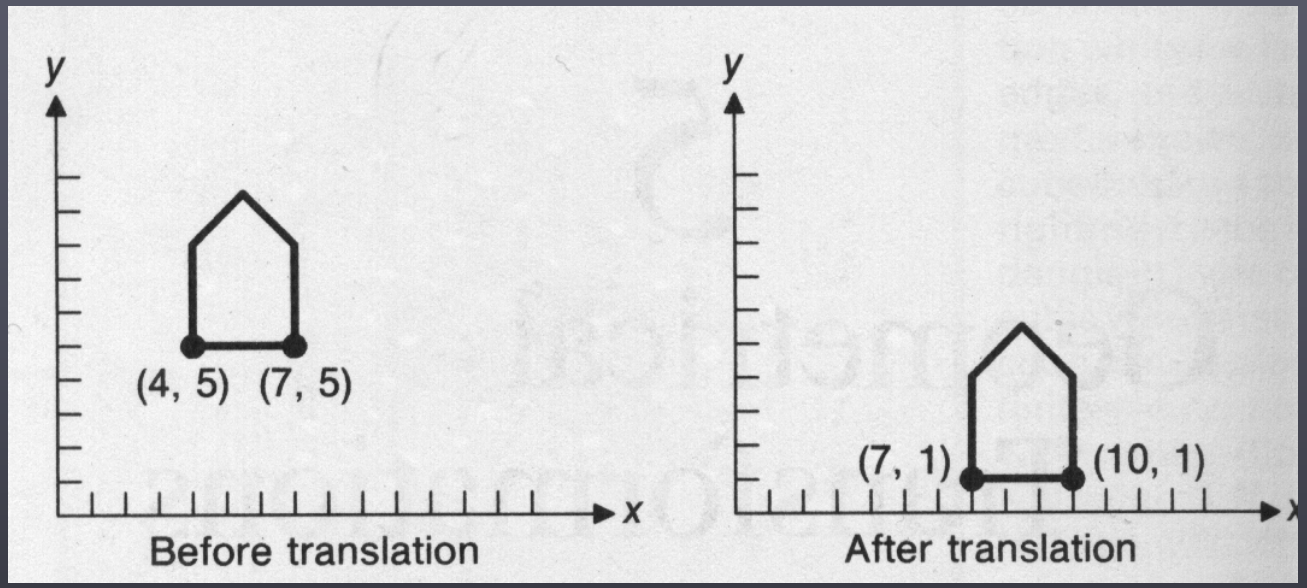


Om vi förutsätter att vi kan rita linjer kan vi återskapa "Asteroids"!
Men vad behöver vi mer?

Transformationer

- Inom datorgrafik är **transformationer** den kanske viktigaste formen av operation.
- De vanligaste transformationerna är **linjära** och kan skrivas som matriser.
- Många är också **affina**, d.v.s. alla punkter på en linje fortsätter att ligga på linjen efter transformationen, parallella linjer bevaras, och förhållandet mellan avstånd bevaras.

Translation

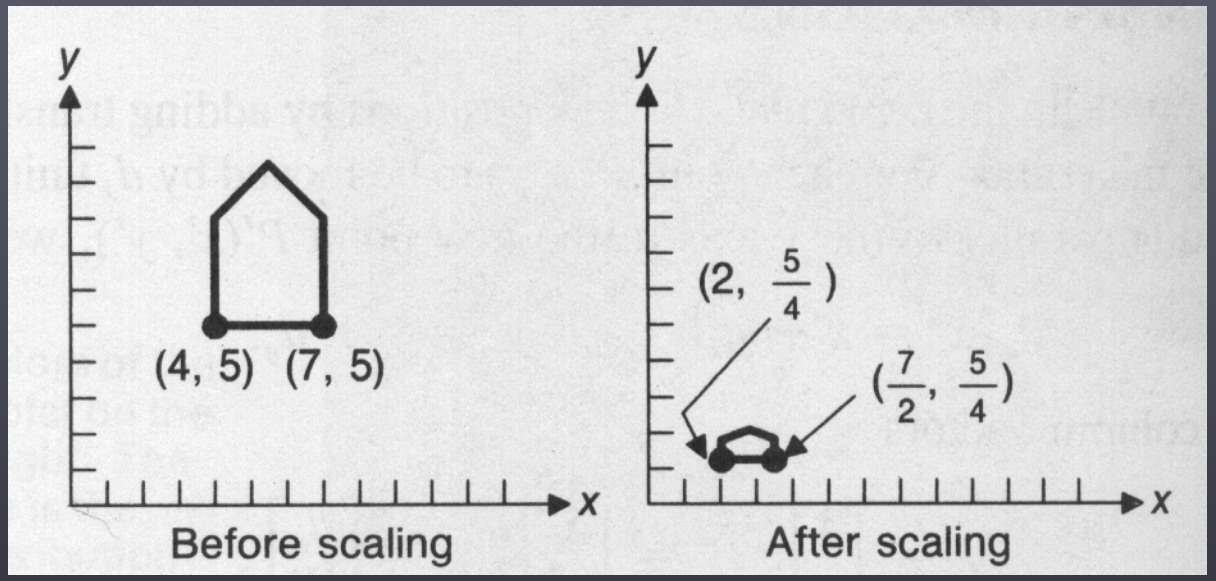


$$x' = x + d_x, \; y' = y + d_y$$

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

$$P' = P + T$$

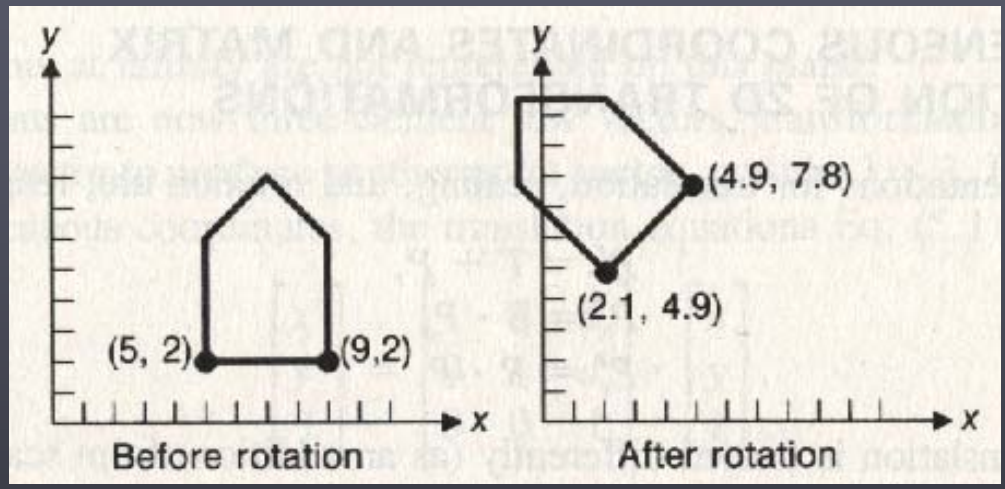
Skalning



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Skalningen sker kring origo.

Rotation



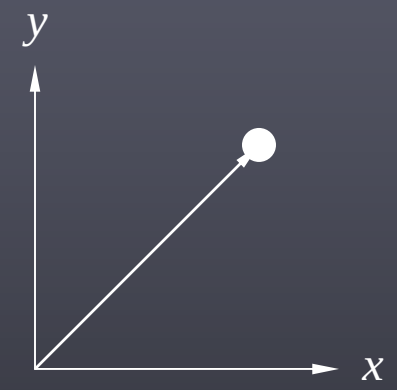
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotationen sker kring origo.

Homogena koordinater

- Det vore bra om vi kunde skriva alla dessa transformationer på ett konsekvent sätt.
- Det funkar om vi inför en tredje koordinat **w**.
- Definition: två punkter är lika om den ena är en multipel av den andra.
- Axiom: Åtminstone en av de tre koordinaterna måste vara skild från 0.
- Vi säger att (x/w, y/w) är de **kartesiska koordinaterna** (cartesian coordinates).
- När w = 0 ligger punkten i oändligheten och vi definierar att (x, y) då representerar en **riktning/vektor**.

Punkter och riktningar



Riktningar (vektorer) och punkter är **olika** geometriska entiteter och existerar oberoende av referenskoordinatsystem!

Men de kan definieras i förhållande till ett sådant.
Och vi behöver ett för att kunna specificera dem med siffror.

Transformationsmatriser

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Skalning

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

I affina transformationer ändras aldrig w-koordinaten.

Kombination av transformationer

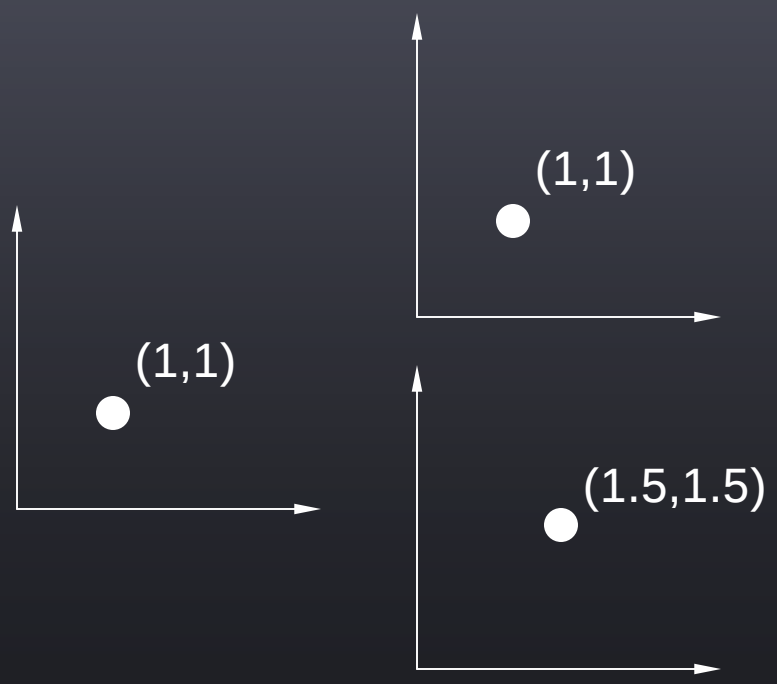
- Vi kan nu kombinera transformationer genom att multiplicera ihop matriserna!
- Men resultatet kan bli olika om man vänder på ordningen!

$$T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

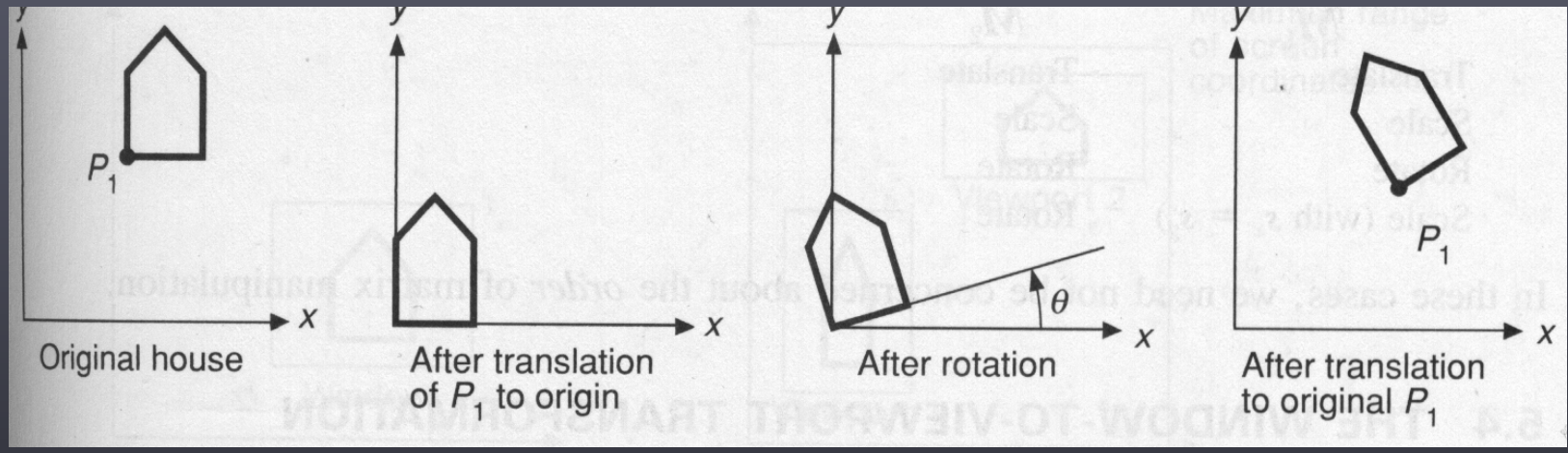
$$(ST_0)P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$(T_0S)P = \begin{bmatrix} 1.5 \\ 1.5 \\ 1 \end{bmatrix}$$



Exempel

Rotera kring en punkt annan än origo:

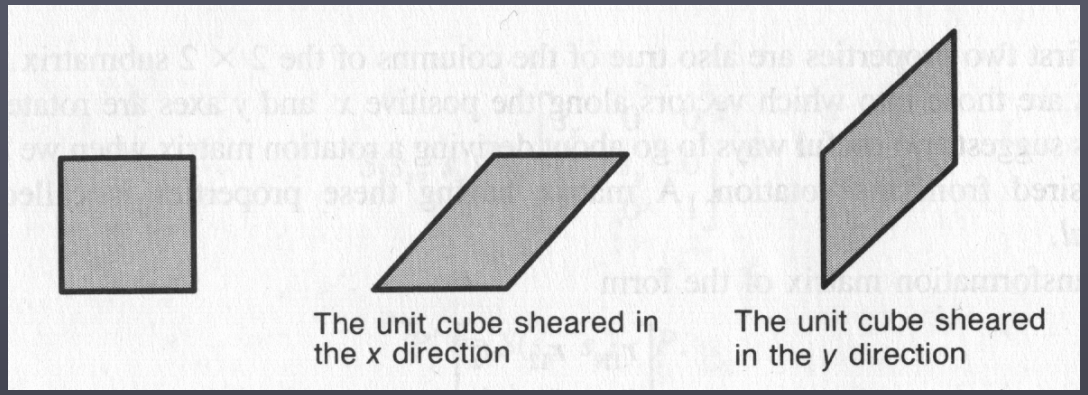


Mer om rotationsmatriser

- Om R är en rotationsmatris är den översta 2x2-delmatrisen **ortogonal**.
- Riktningsvektorerna $(r_{11},r_{21},0)$ och $(r_{12},r_{22},0)$ är de två vektorer som x- resp. y-axeln är roterade till!

$$R = \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

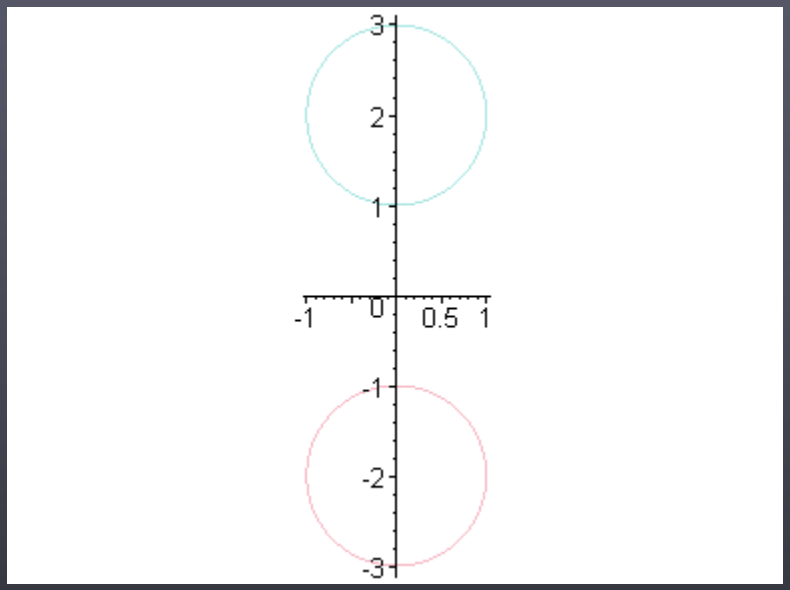
Skevning (shear)



$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Spegling (reflection)



$$RE = \begin{bmatrix} 2u_x^2 - 1 & 2u_x u_y & 0 \\ 2u_x u_y & 2u_y^2 - 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Där (u_x, u_y) är riktningsvektorn (med längd 1) man speglar igenom.

Ickelinjära transformationer

- Förekommer inte lika ofta som linjära, men är inte alls ovanliga.
- Exempel: Environment mapping.

$$s = \frac{R_x}{2\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}} + \frac{1}{2}$$

$$t = \frac{R_y}{2\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}} + \frac{1}{2}$$

Transformationer av hörn

- Det vore i praktiken omöjligt i 3D-grafik att transformera varje pixel för sig.
- Om vi använder linjära transformationer räcker det dock att transformera hörnen och binda samman dem med linjer!
- Man interpolerar sedan data definierad i hörnen linjärt över primitiverna.
- Om det är färg man interpolerar brukar det kallas **Goraud shading**.

I ett spel som "1942" används inte linjer.
Vilka element består grafiken av? Hur funkar det?

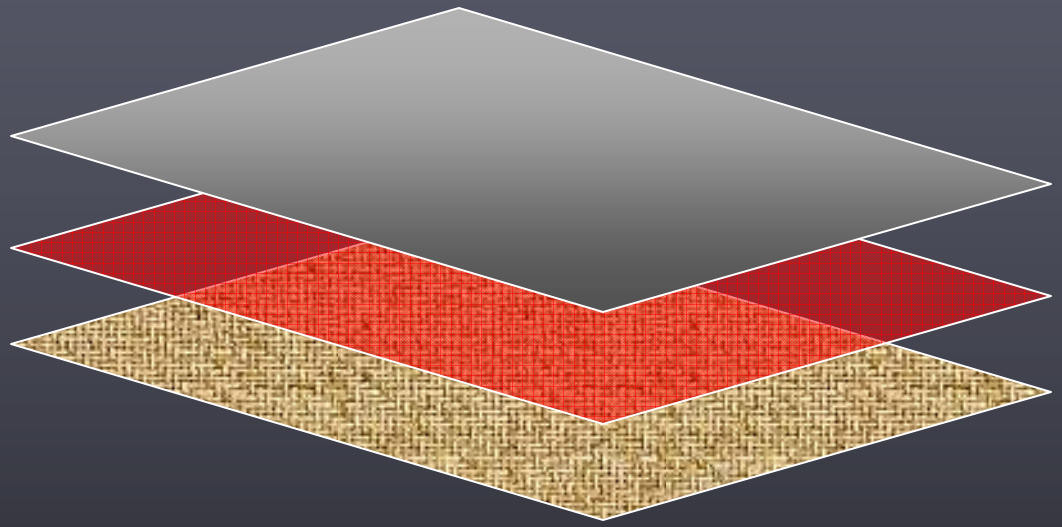
Compositing

Compositing

Georges Méliès, tidigt 1900-t.

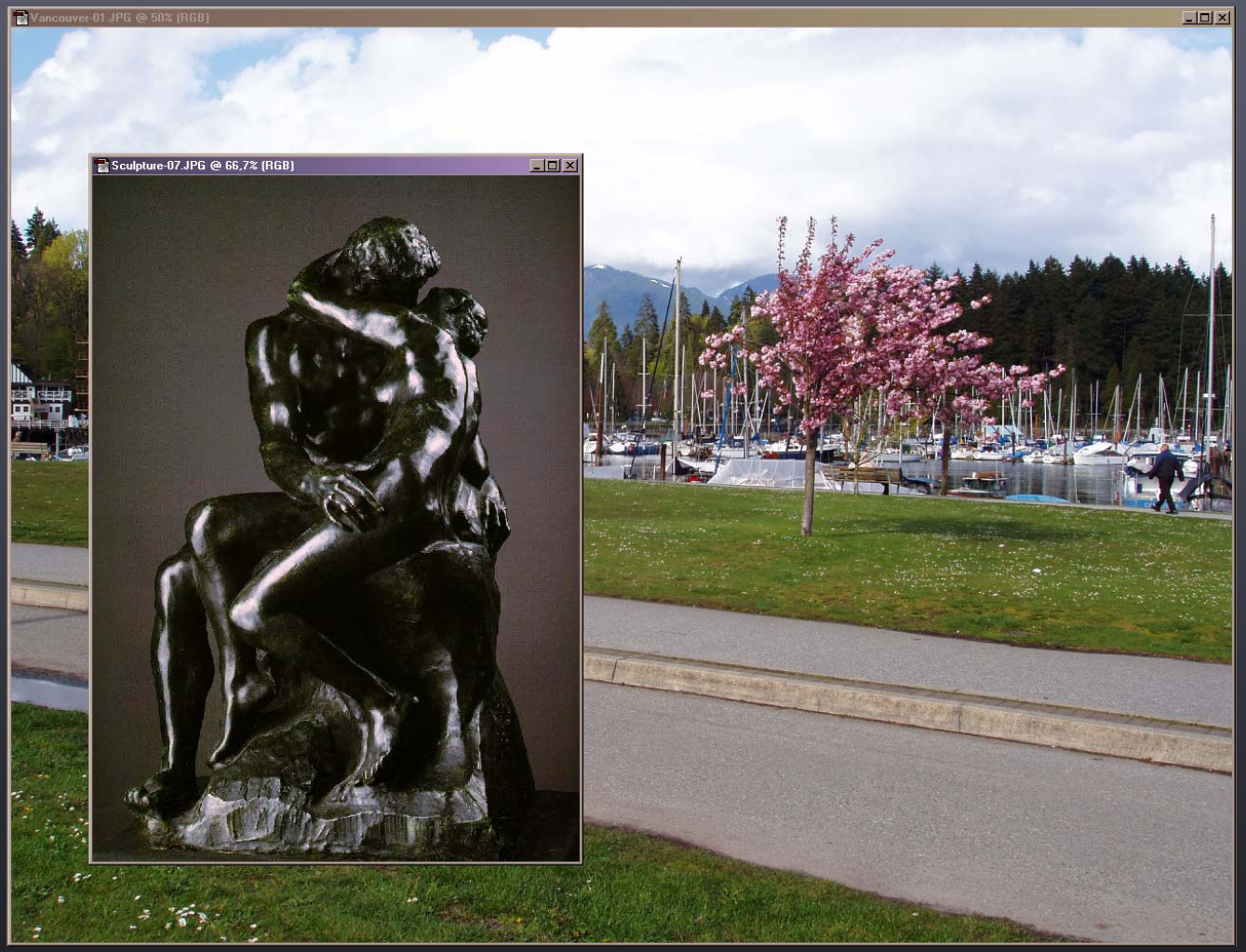
Compositing

Lager

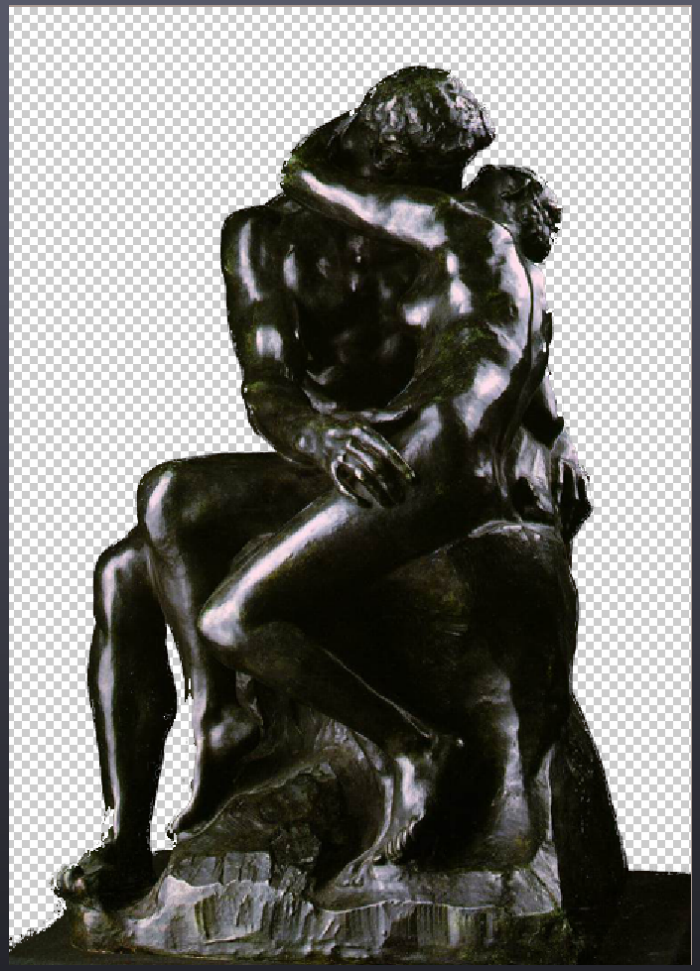


Bildbuffrar som kombineras ihop pixel för pixel nedifrån och upp.
För varje nytt lager definierar man en funktion som beskriver hur informationen från föregående lager ska kombineras ihop med aktuellt lager.

Exempel

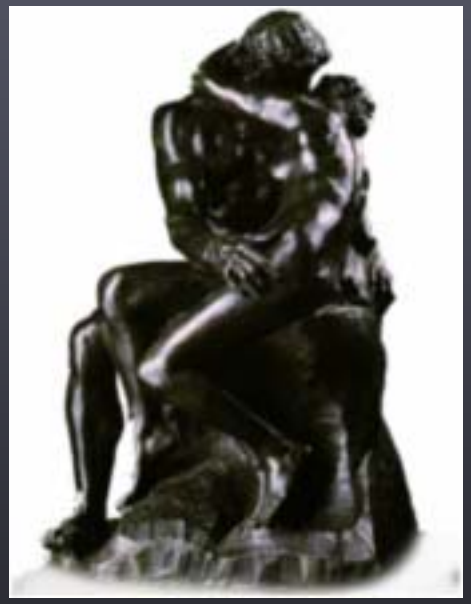


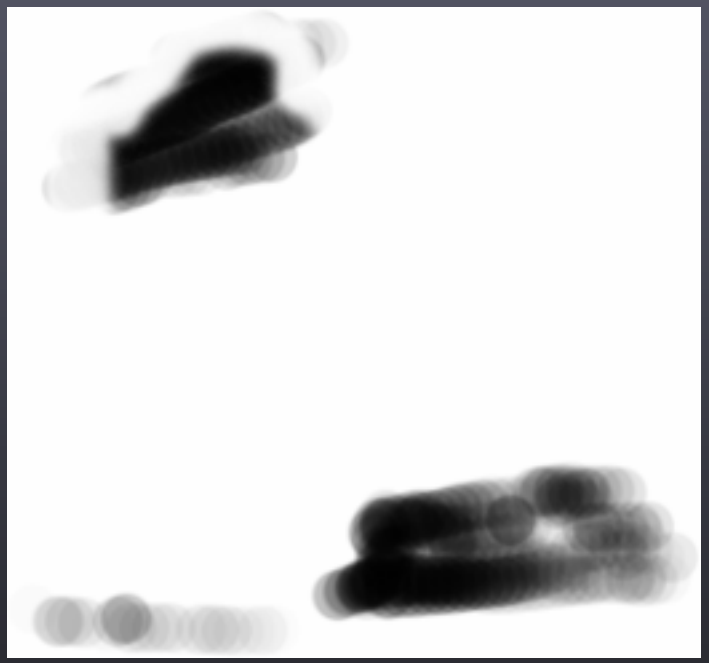
Exempel



Exempel

Exempel





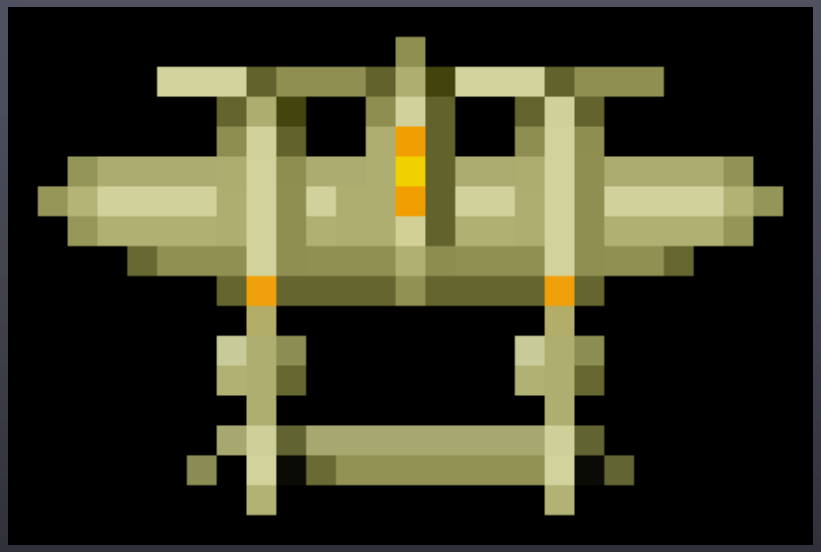
Sprites

Monkey Island 3 (PC)

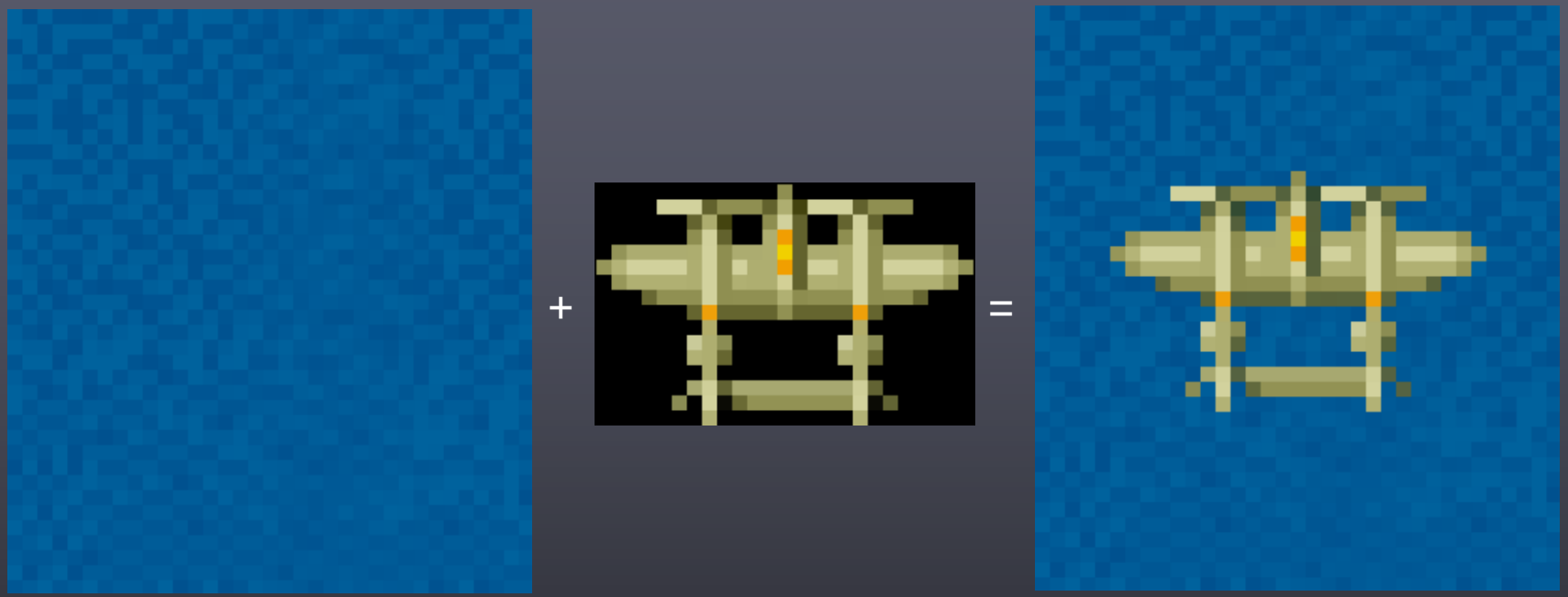
1942 (Arkad)

Sprites

- **Sprite** är en term från 80-talet
- I färgpaletten för en 8-bitarsbild bestämmer man att en av färgerna ska vara genomskinlig
- Idag använder man alpha-kanalen istället



Sprites



Bilden byggs upp i lager, nerifrån och upp.
I många äldre datorer fanns en operation för att snabbt "sprite"-kopiera från minne till framebuffer, s.k. **bitblt**.
Vissa maskiner kunde skala och rotera sprites vid utritningen.
Ibland fanns stöd för kollisionsdetektion sprites emellan.

Sprites

http://www.nes-snes-sprites.com/DonkeyKongCountryNew.html

Det blir **MÅNGA** sprites, ibland en för varje animationsruta!
(Donkey Kong Country för SNES hade över tusen, t.ex.)
Duktiga spriteteckнare återanvände bildrutor.

Om bitblt-operationen kunde spegla bilderna
sparade man mycket minne.

Lager och sprites

Monkey Island 3 (PC)

Pseudo-3D

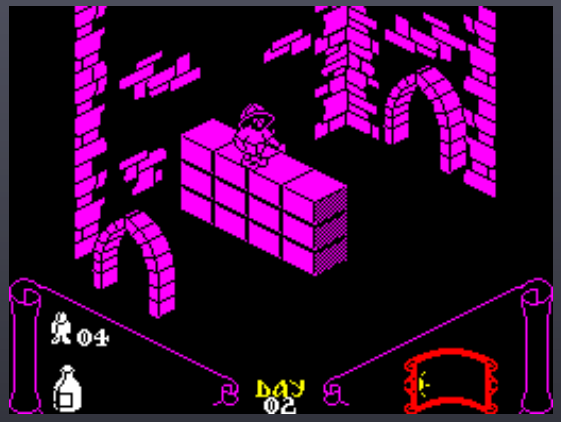
Out Run (Arkad)

Parallax-scrolling

Moon Patrol (Arkad)

Isometriska spel

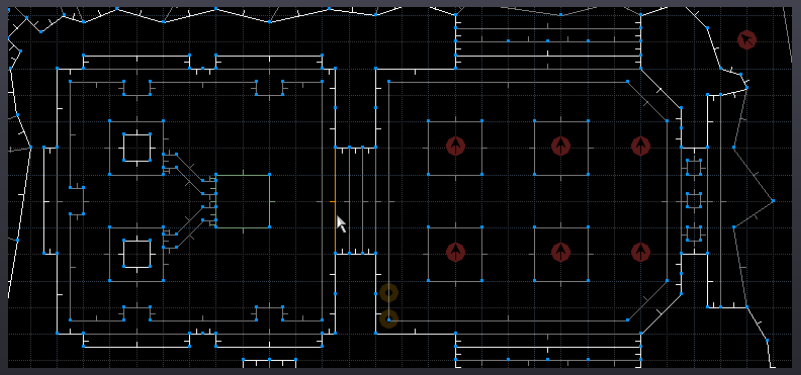
Q*Bert (Arkad)



Knight Lore (ZX Spectrum)

SimCity 2000 (PC)

2.5D



Doom

En **billboard** är en polygon som alltid vänder "framsidan" åt kameran.

2D-grafik i Java

```
import java.awt.*;
import javax.swing.*;

public class GrafikTest extends JFrame {
	public void paint(Graphics g) {
		g.drawString("Hello", 10, 50);
	}

	public static void main(String[] args) {
		GrafikTest f = new GrafikTest();
		f.setTitle("GrafikTest");
		f.setSize(300, 300);
		f.setVisible(true);
	}
}
```

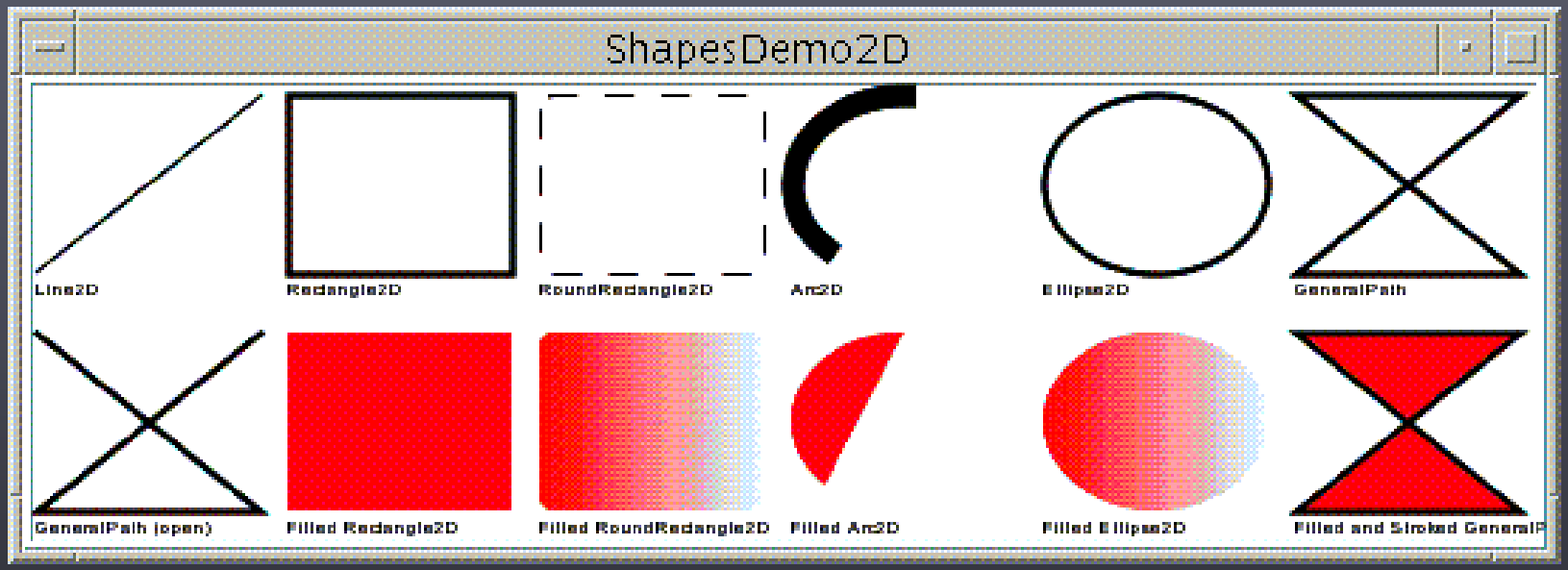
2D-grafik i Java

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class Java2D extends JFrame {
        public void paint(Graphics g) {
                Graphics2D g2 = (Graphics2D) g;
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                    RenderingHints.VALUE_ANTIALIAS_ON);
                BasicStroke bs = new BasicStroke(10);
                g2.setStroke(bs);
                GeneralPath p = new GeneralPath();
                p.moveTo(50, 50);
                p.lineTo(150, 50);
                p.lineTo(20, 100);
                p.closePath();
                g2.draw(p);
        }

        public static void main(String[] args) {
                Java2D f = new Java2D();
                f.setTitle("Java2D");
                f.setSize(300, 300);
                f.setVisible(true);
        }
}
```

2D-grafik i Java



Transformationer i Java2D

```
GeneralPath p = new GeneralPath();
p.moveTo(50, 50);
p.lineTo(150, 50);
p.lineTo(20, 100);
p.closePath();
g2.draw(p);

AffineTransform t =
AffineTransform.getTranslateInstance(100, 150);
t.rotate(-Math.PI/3.0);
AffineTransform old = g2.getTransform();
g2.transform(t);
g2.draw(p);
g2.setTransform(old);
```

Mer om Java2D

- Det finns möjlighet att använda lager och masking i Java2D.
- Man kan också kombinera ihop primitiver automatiskt med boolska operationer.
- Stöd för att ladda och rita bitmaps/bilder finns också.