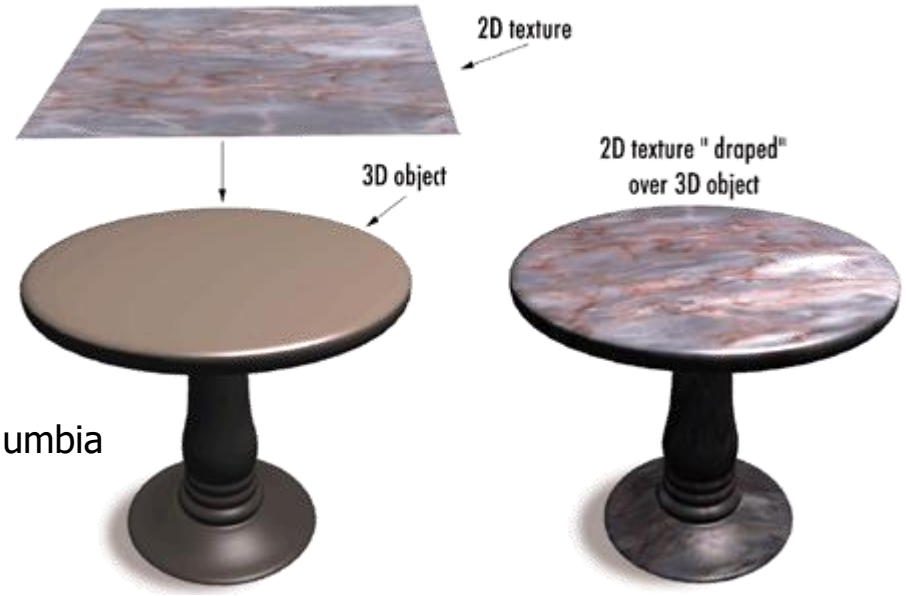
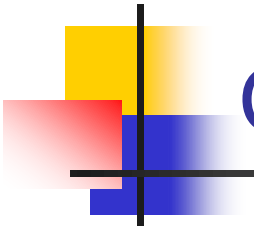
Texture mapping

Robert G. Belleman
Universiteit van Amsterdam

Chapter 11 Marschner & Shirley, FCG2/4.

Met materiaal van
- Kurt Akeley, Stanford University en
- Tamara Munzner, University of British Columbia
- e.a.



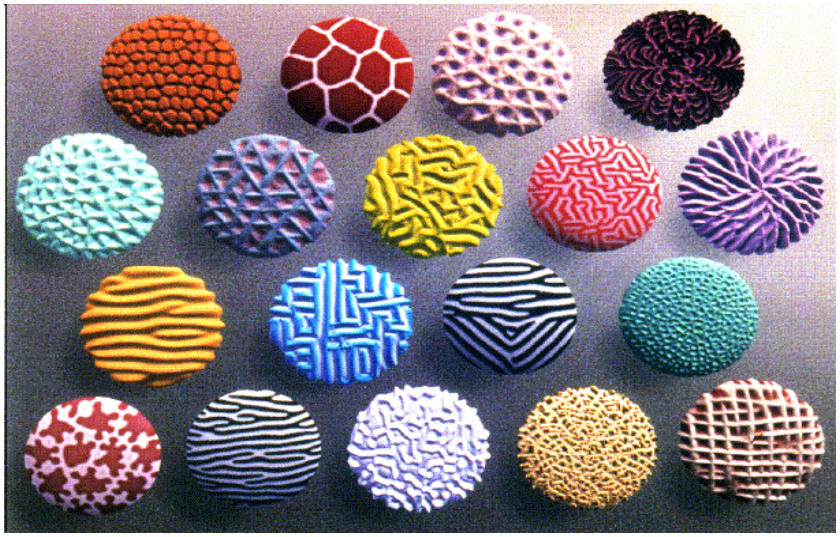


Overzicht

- Texture mapping
  - Waarom?
  - Hoe gebruik je het?
  - Hoe werkt het?
- Toepassingen:
  - Bump mapping
  - Displacement mapping
  - Environment mapping
  - Direct volume rendering

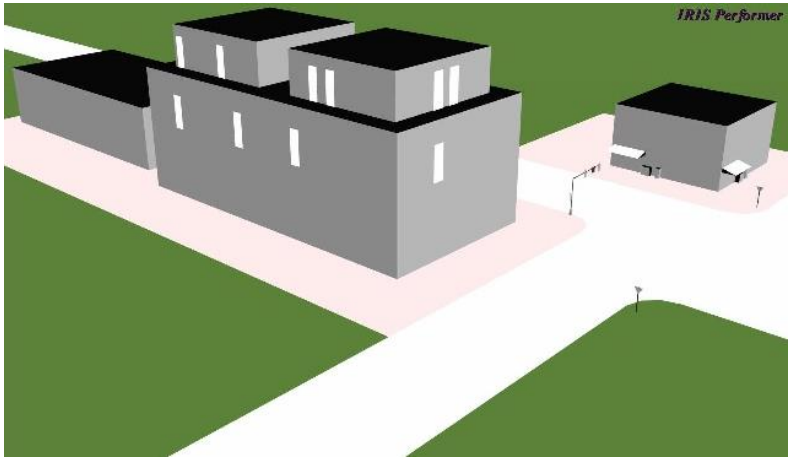
Texture mapping: waarom?

- Belichtingsmodellen zorgen voor uniforme belichting
  - Onnatuurlijke, saaie, schone, “plastic” objecten
- Echte objecten zijn niet uniform gekleurd
  - Kleurvariaties, onregelmatig oppervlak: “textuur”
- Texture mapping wordt gebruikt om geometrische details te benaderen



Texture mapping: waarom?

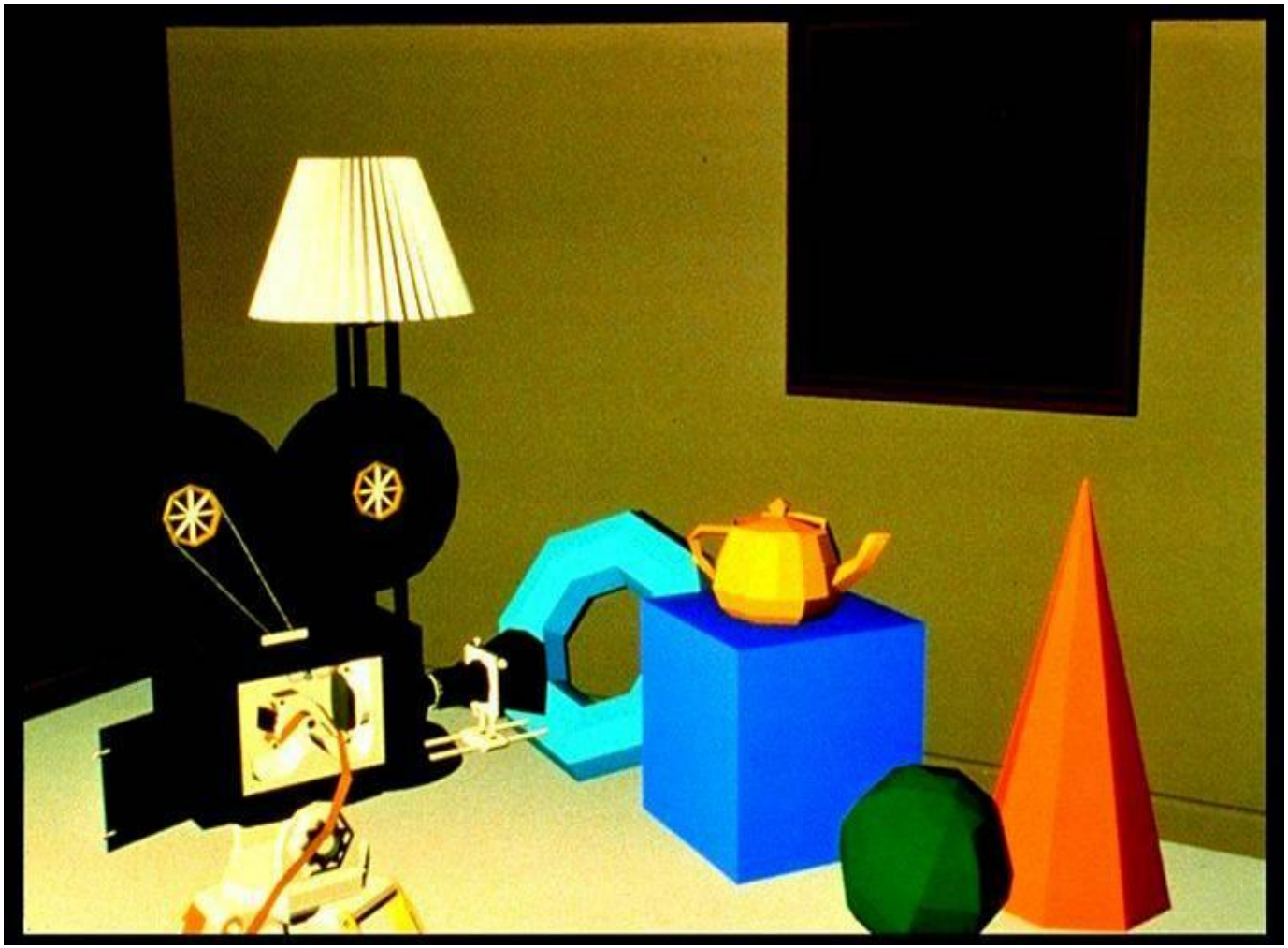
- Om objecten realistischer te maken
  - Als eenvoudige belichtings-modellen niet voldoende zijn
- Om complexe geometrie te benaderen
  - Onregelmatige geometrie wordt benaderd door een plaatje



Texture mapping

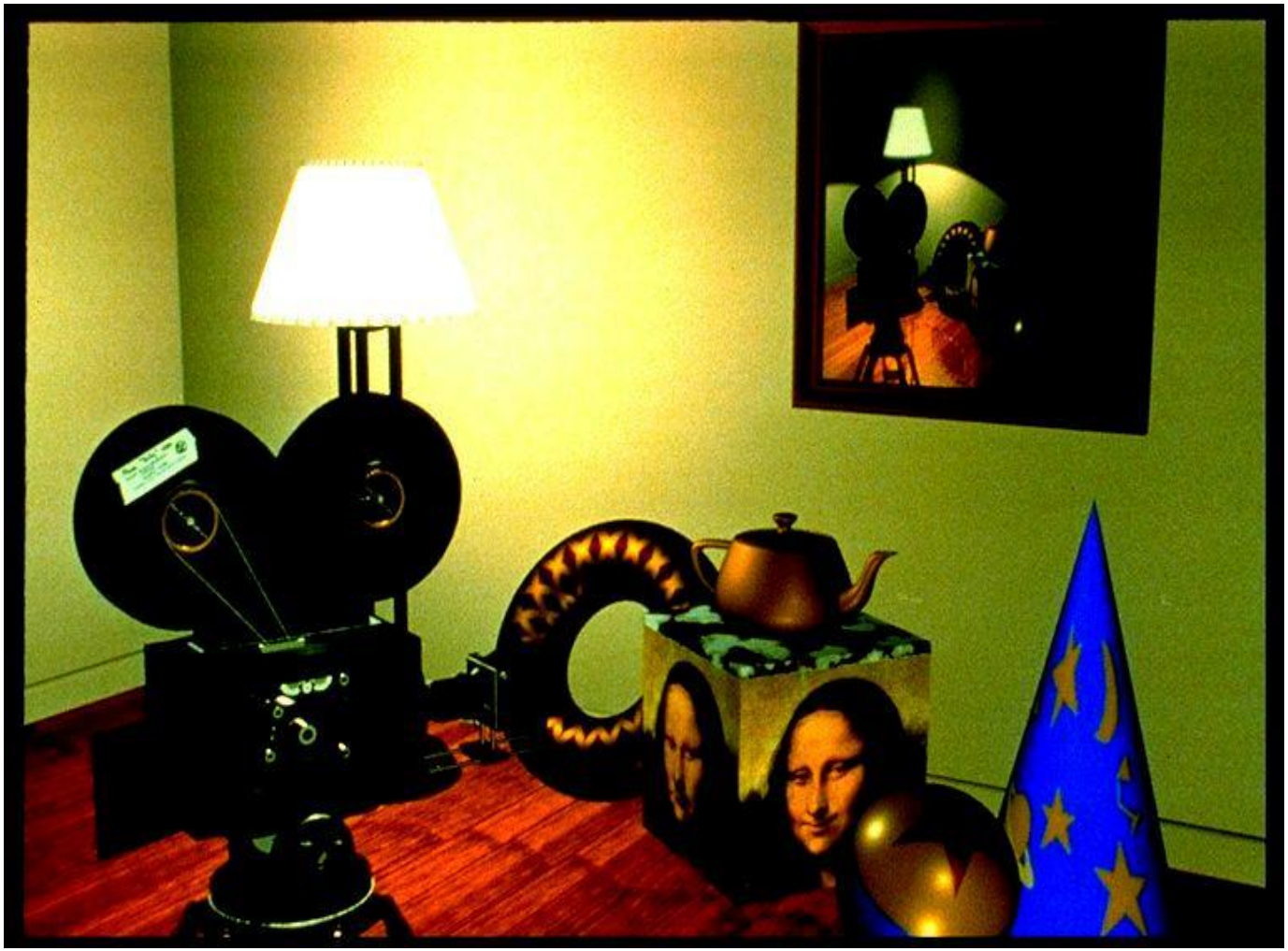
- Plak plaatjes op driehoeken, lijnen, punten en andere plaatjes
- Bij rasterizatie wordt de kleur van een pixel op een object mede bepaald a.d.h.v. een plaatje (i.p.v. alleen belichting)
- Twee benaderingen:
  - Oppervlakte textures (2D textures)
  - Volumetrische textures (3D textures)

No texture mapping



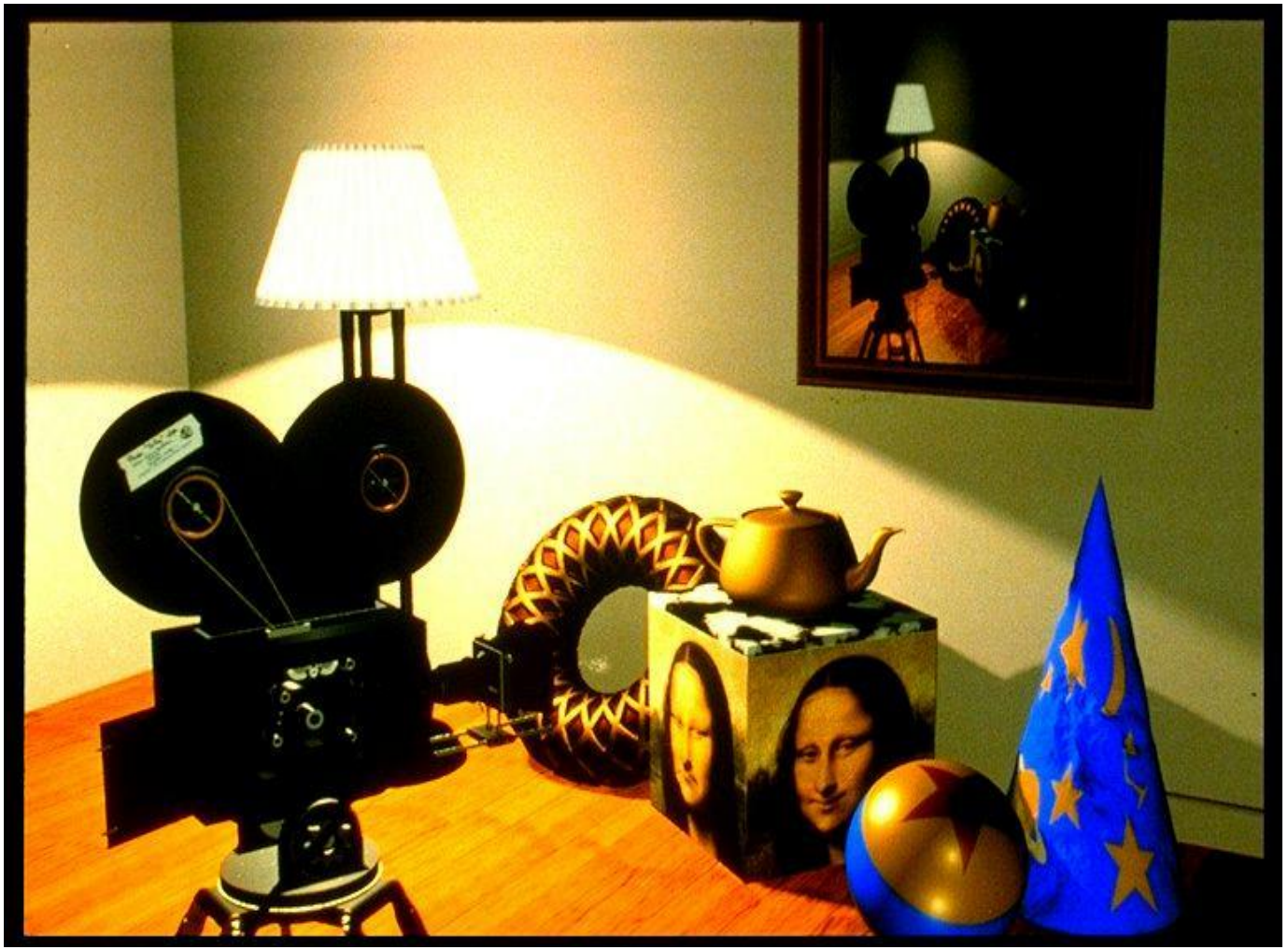
Source: Foley, van Dam et al., Computer Graphics: Principles and Practice

Texture Mapping



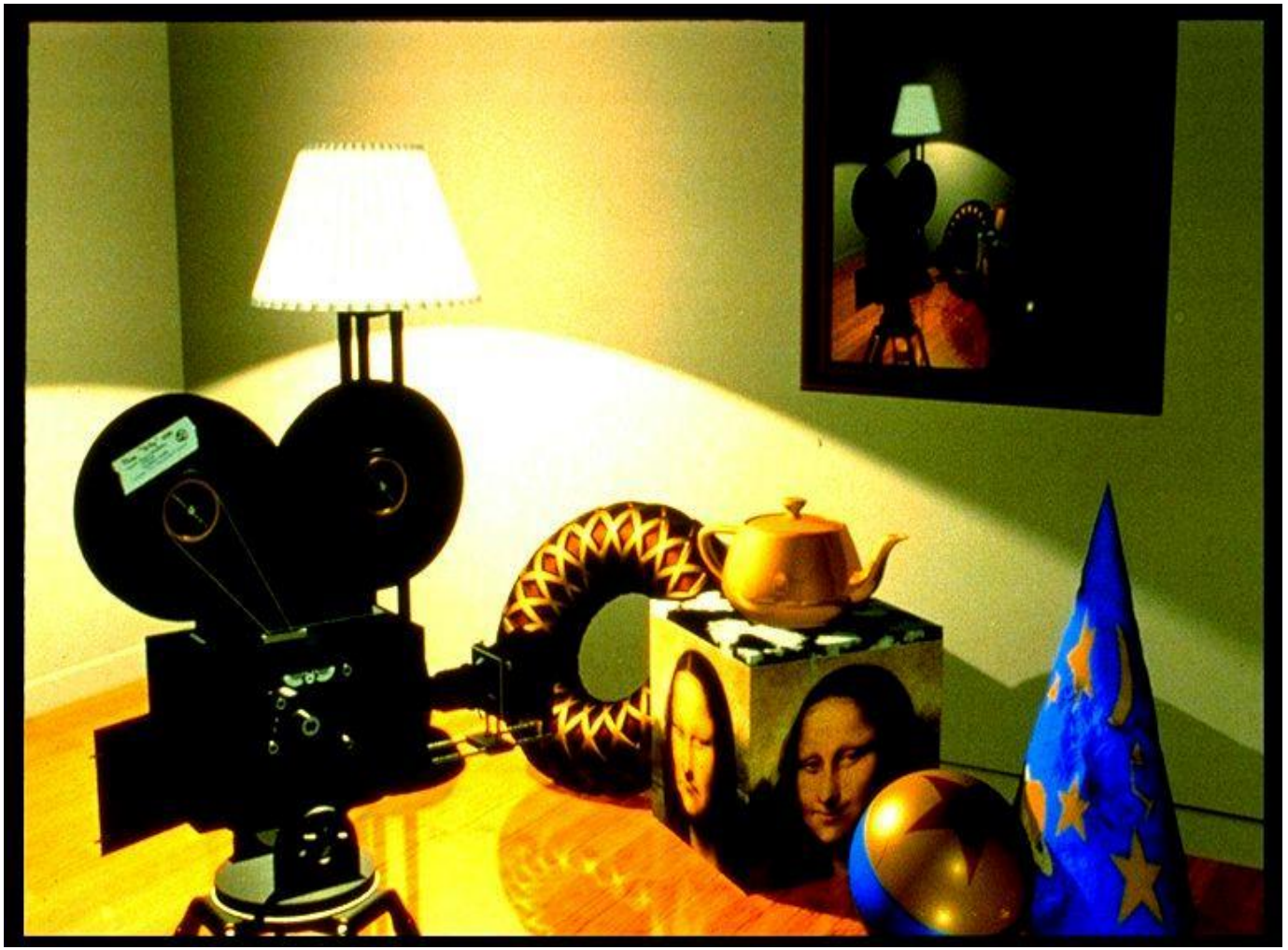
Source: Foley, van Dam et al., Computer Graphics: Principles and Practice

Displacement Mapping

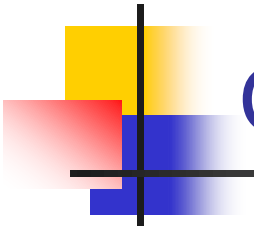


Source: Foley, van Dam et al., Computer Graphics: Principles and Practice

Reflection Mapping



Source: Foley, van Dam et al., Computer Graphics: Principles and Practice



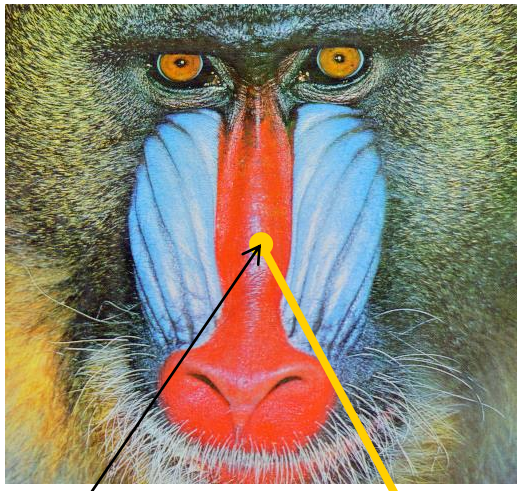
Grondbeginselen

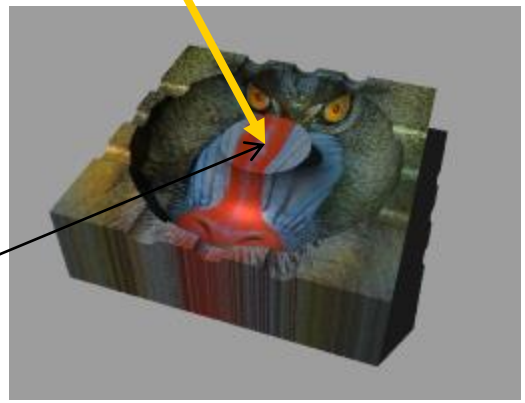
Voor texture mapping heb je het volgende nodig:

1. Het texture image: een plaatje
2. De mapping van geometrie naar coordinaten in het plaatje
3. Het sampling mechanisme
4. De toepassing van de resulterende waarde(n)

Texture coordinaten

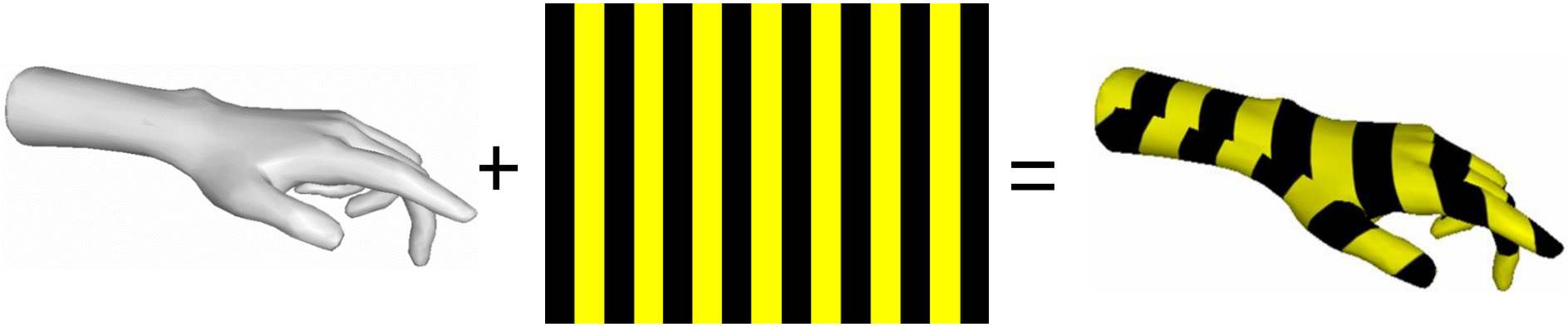
- Texture image is een 2D array van kleurwaarden (“texels”), bv:
  - een plaatje
  - een functie/procedure (“procedural texture”)
- Definieer voor elk punt in een object (x,y,z,w) een texture coordinaat (s,t) in het plaatje
  - Voor elk pixel worden tussenliggende texels (s,t) opgezocht door interpolatie
  - De kleurwaarde in de texture wordt gebruikt om de kleur van het oppervlak te veranderen
    - Of een andere oppervlakte eigenschap
  - Opgegeven door de programmeur:

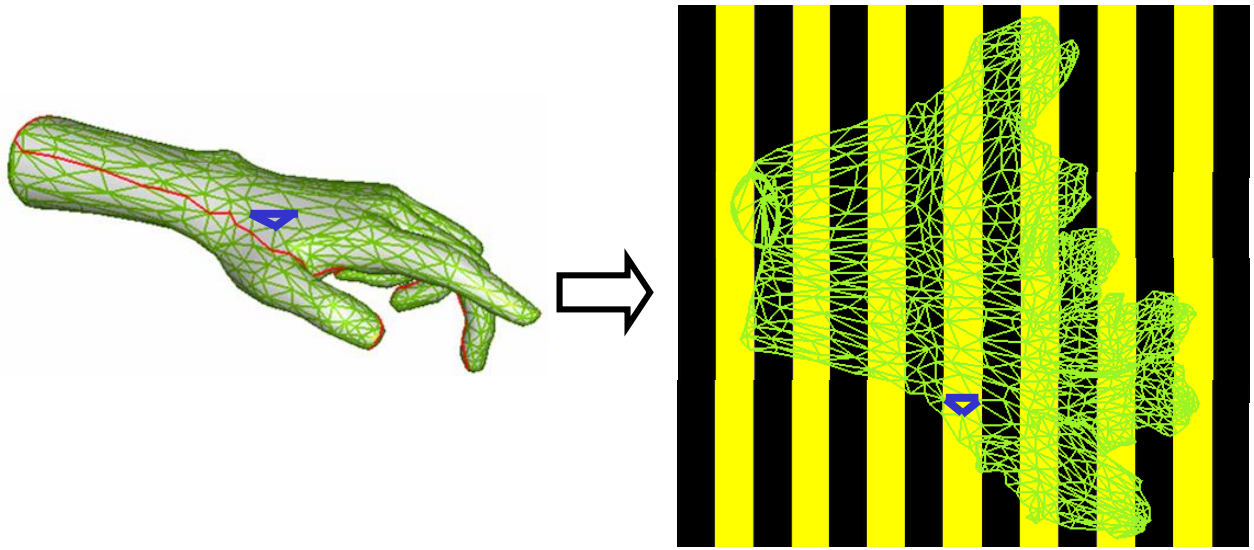




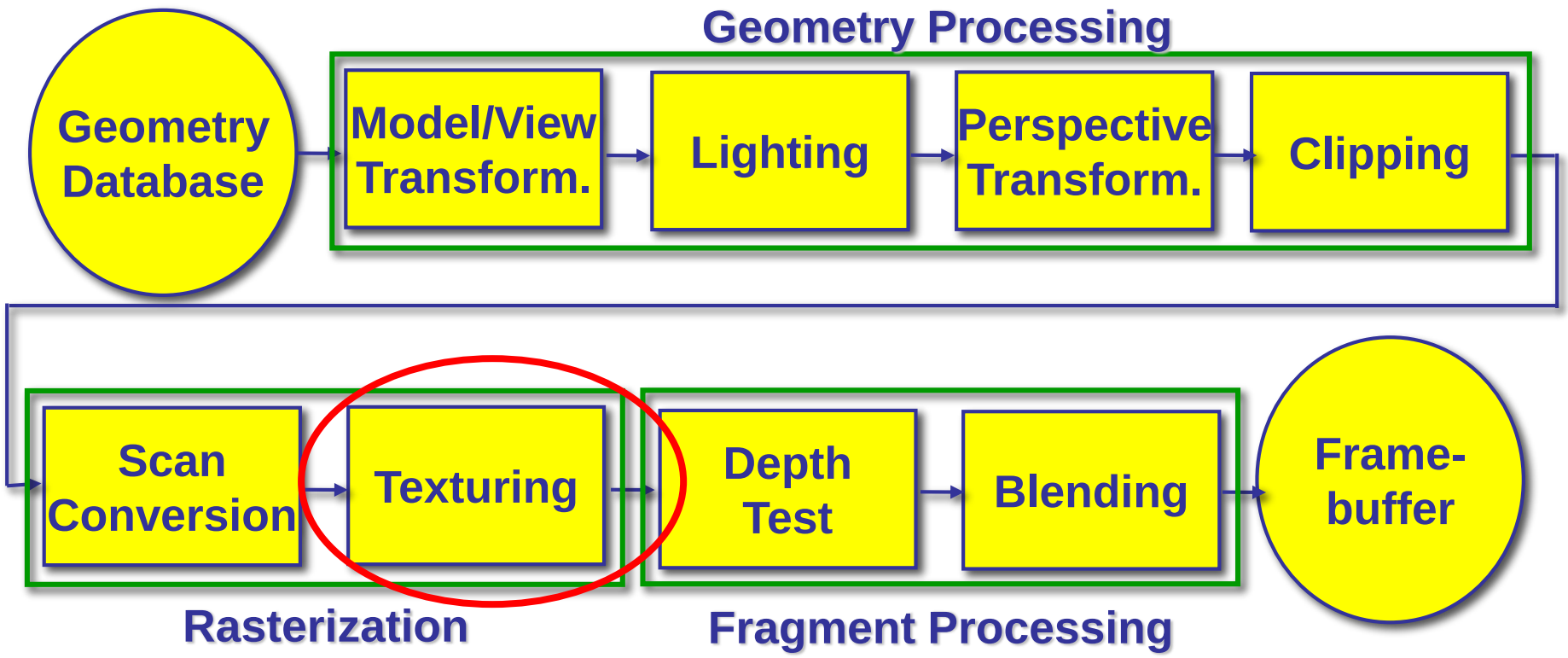
```
glTexCoord2f(s,t)
glVertexf(x,y,z,w)
```

Voorbeeld

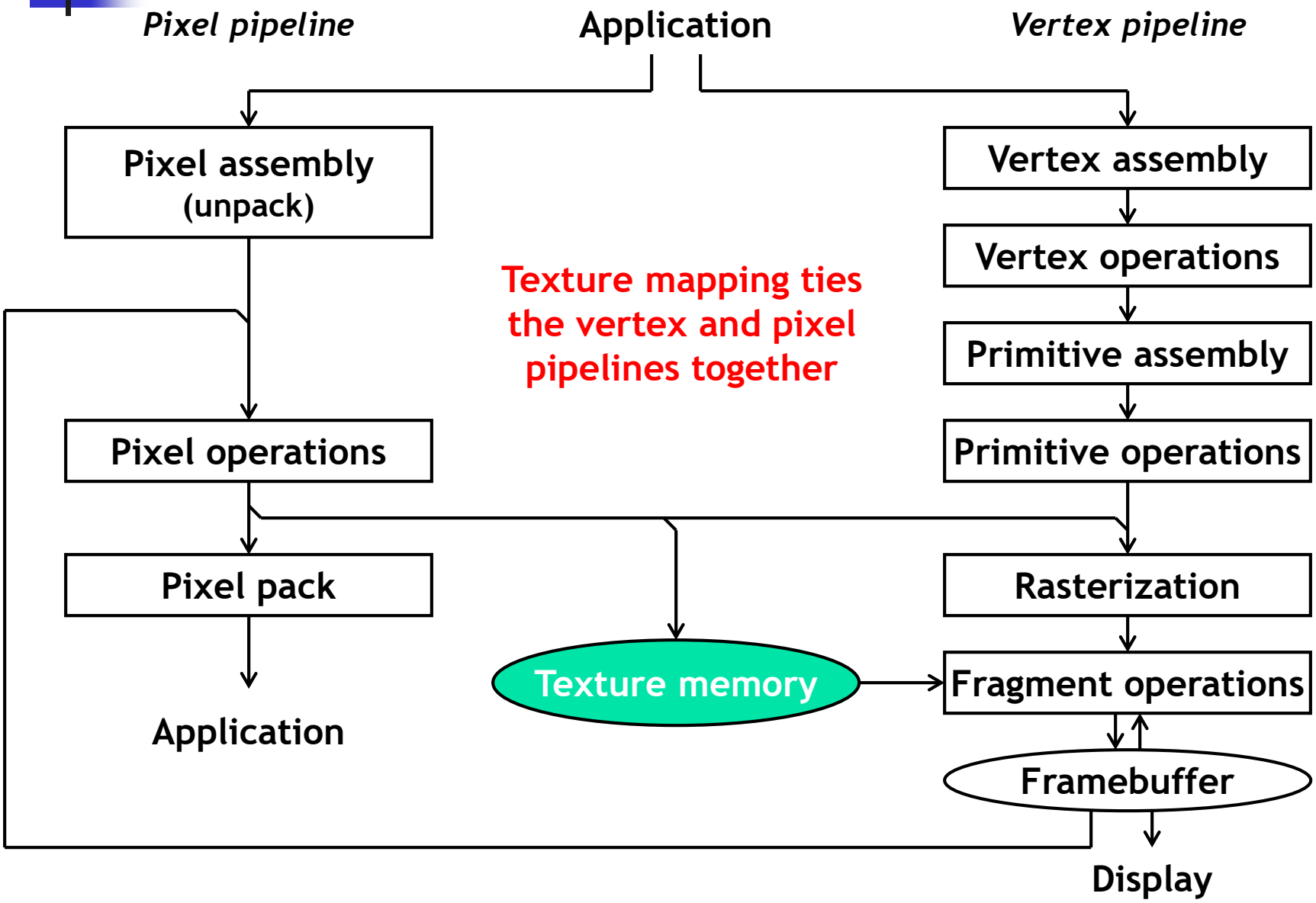




Rendering Pipeline

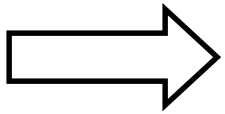


Voorbeeld: complete OpenGL pipeline

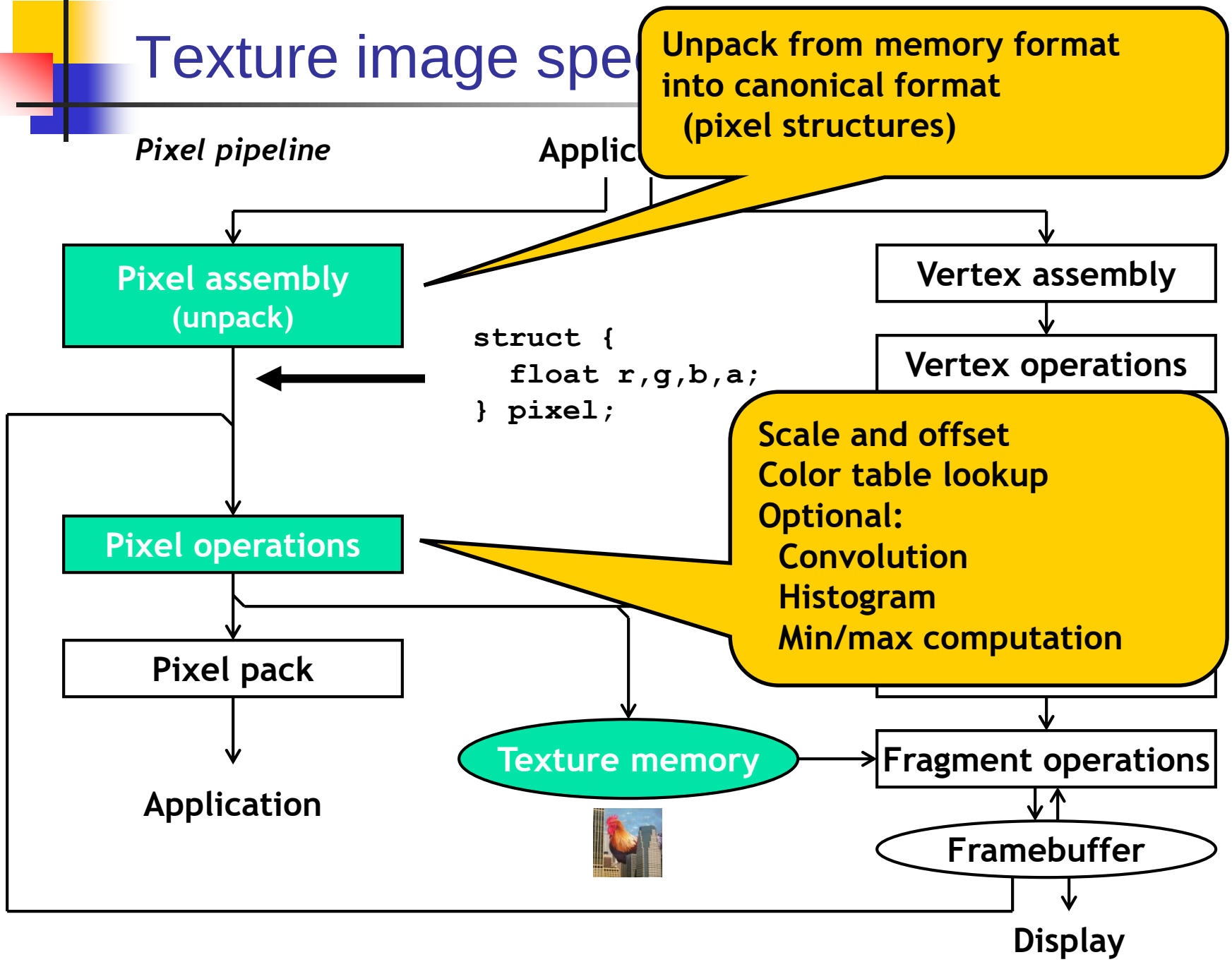


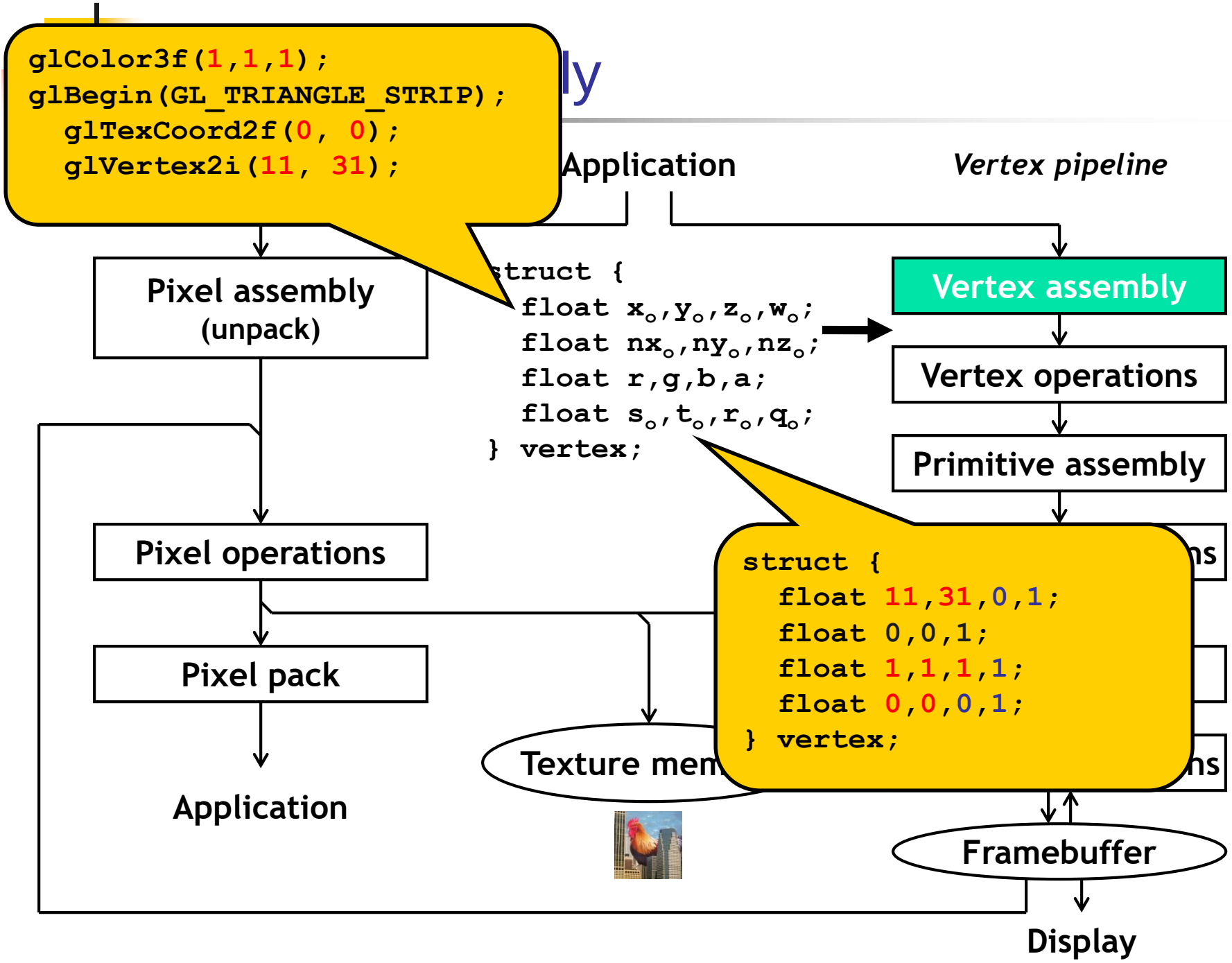
Textured quad in OpenGL

```
glGenTextures(1, &i);          // create 1 texture object
LoadTexture("rooster", i);     // load file into texture object
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, i);
glClearColor(1, 1, 1, 1);      // white
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glOrtho(0, 100, 0, 100, -1, 1);
glColor3f(1, 1, 1);            // white
glBegin(GL_TRIANGLE_STRIP);
  glTexCoord2f(0, 0);          glVertex2i(11, 31);
  glTexCoord2f(0, 1);          glVertex2i(37, 71);
  glTexCoord2f(1, 0);          glVertex2i(91, 38);
  glTexCoord2f(1, 1);          glVertex2i(65, 71);
glEnd();
glFlush();
```

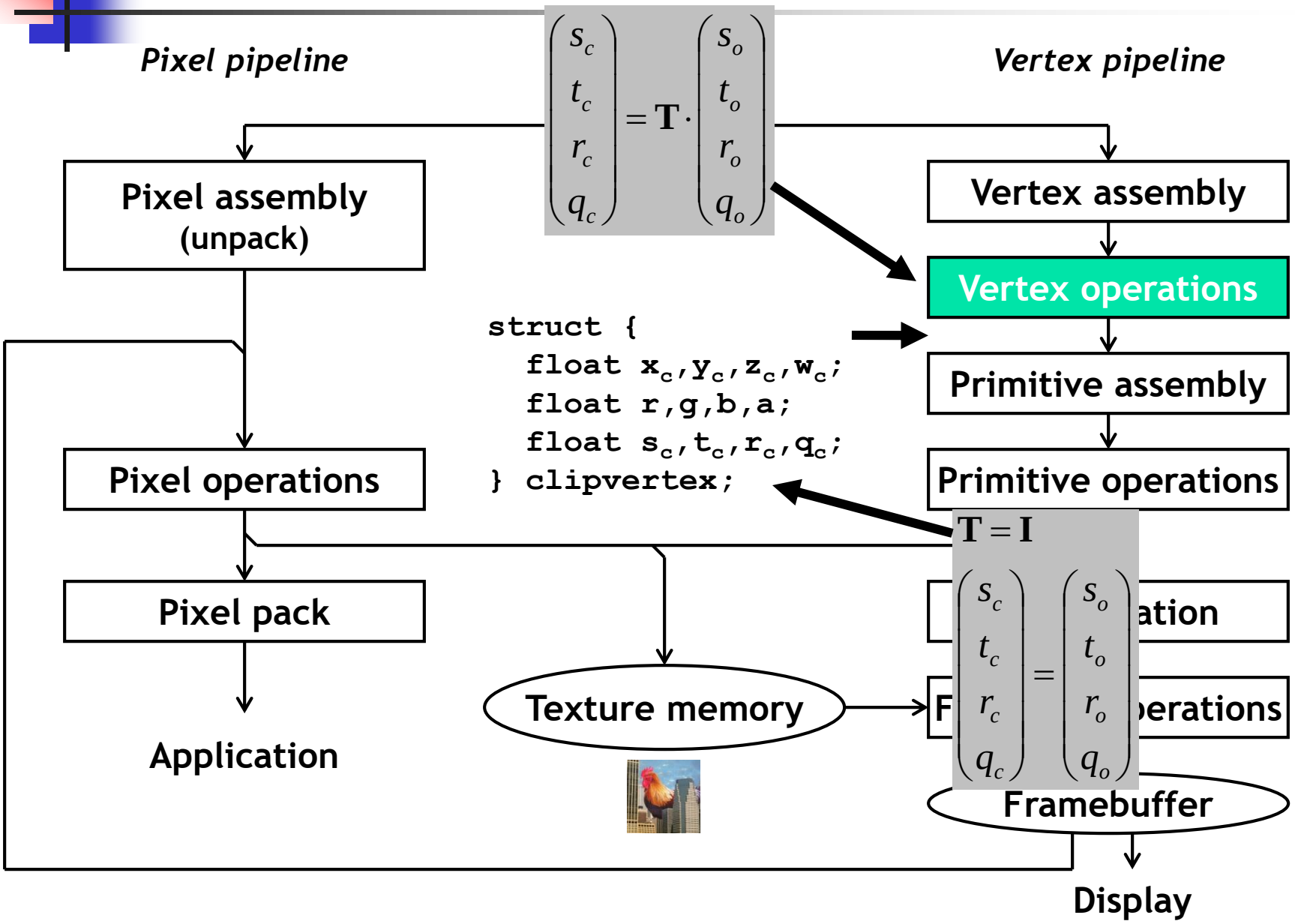


Texture image spe

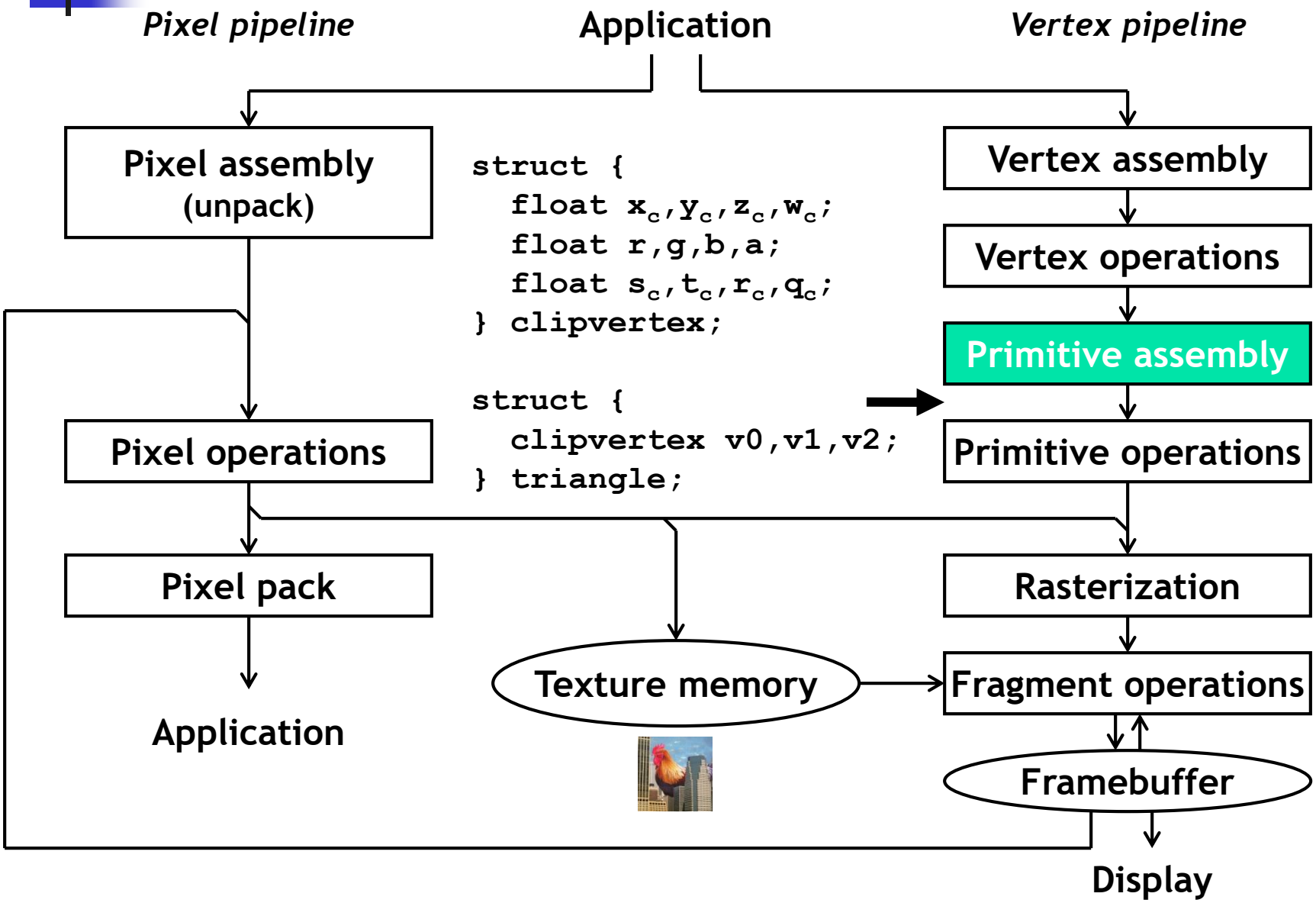



glColor3f(1,1,1);
glBegin(GL_TRIANGLE_STRIP);
glTexCoord2f(0, 0);
glVertex2i(11, 31);
Application
Vertex pipeline
Pixel assembly (unpack)
struct {
float x_o,y_o,z_o,w_o;
float nx_o,ny_o,nz_o;
float r,g,b,a;
float s_o,t_o,r_o,q_o;
} vertex;
Vertex assembly
Vertex operations
Primitive assembly
Pixel operations
struct {
float 11,31,0,1;
float 0,0,1;
float 1,1,1,1;
float 0,0,0,1;
} vertex;
Pixel pack
Texture mem
Application
Framebuffer
Display

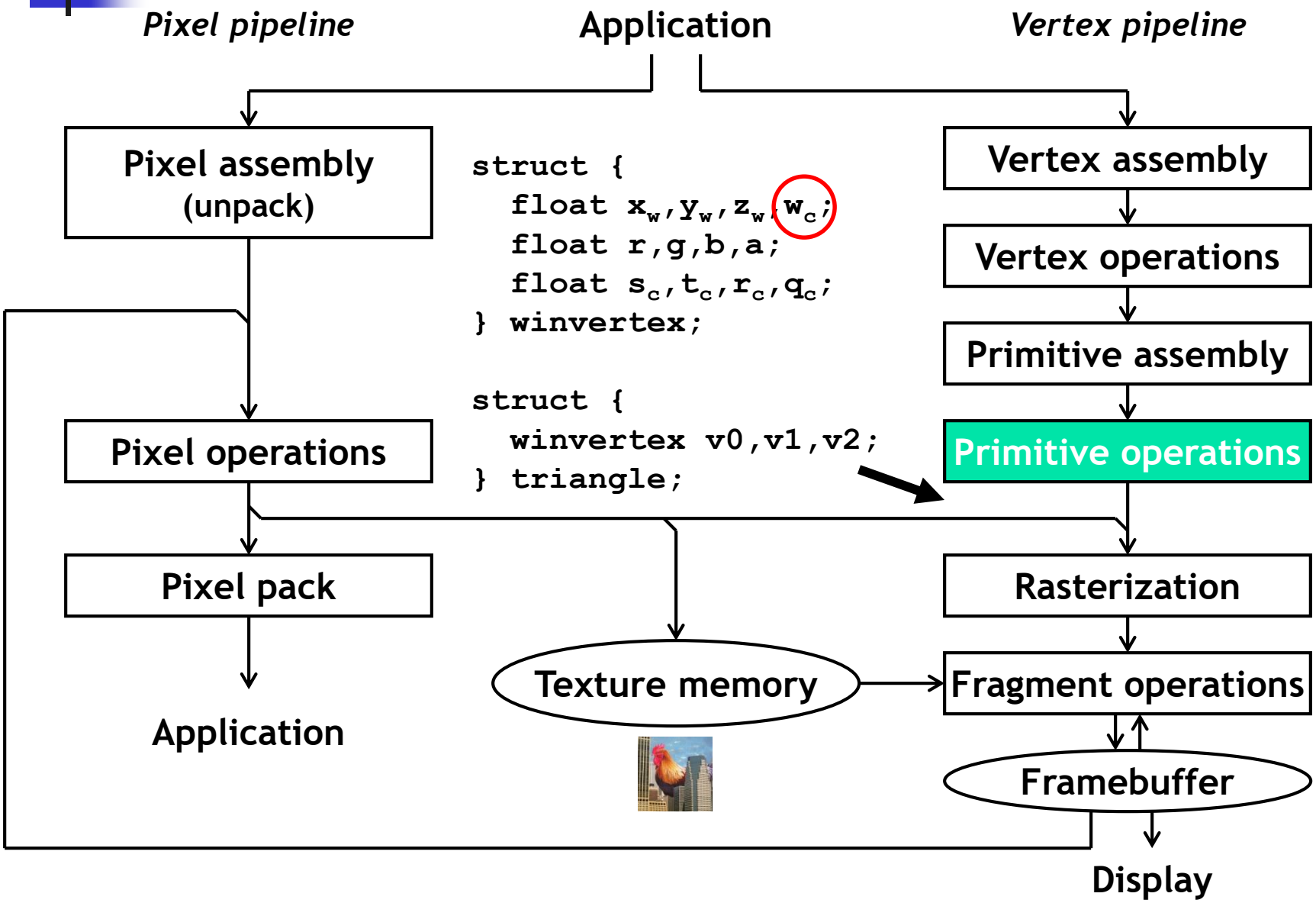
Vertex operations



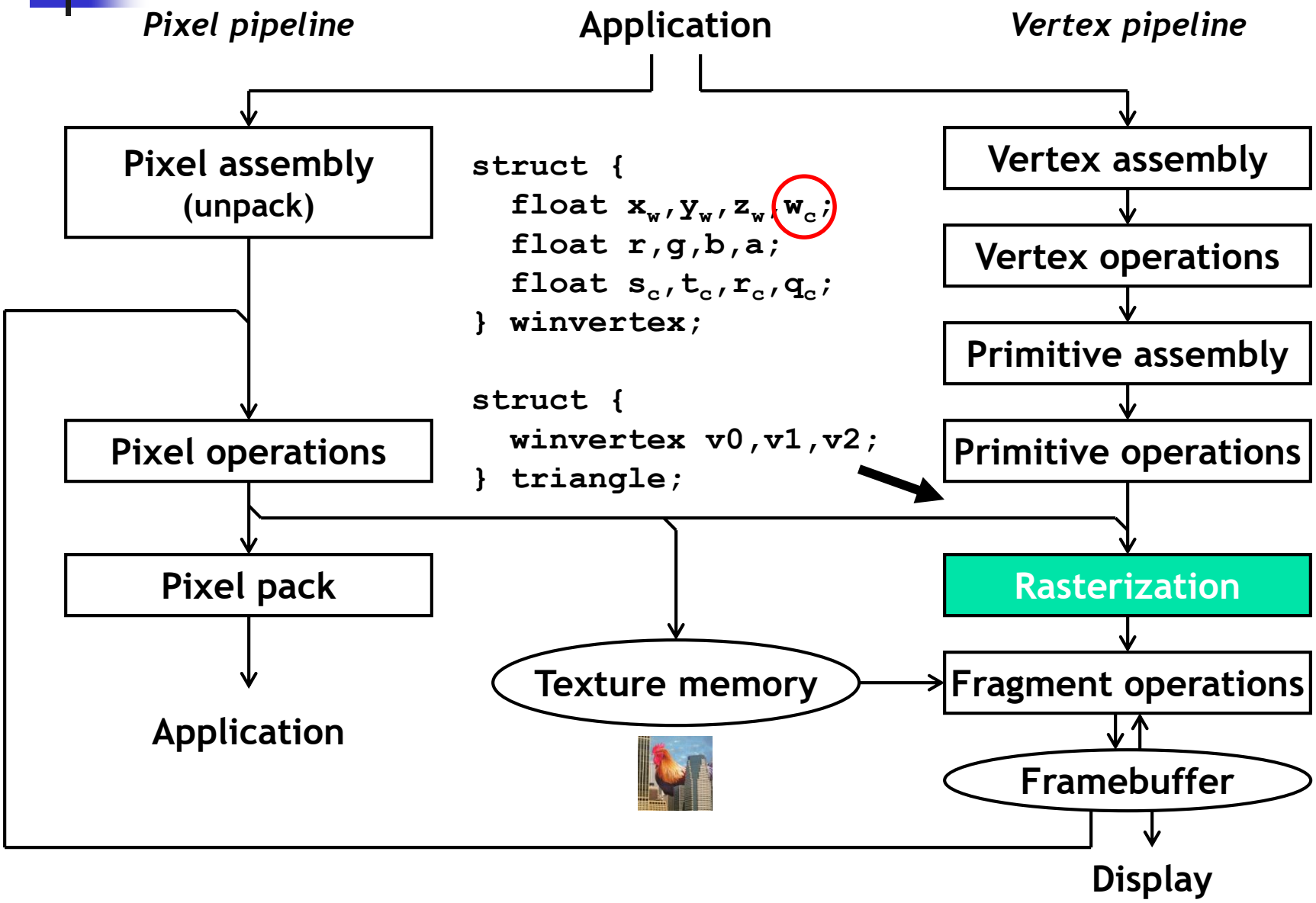
Primitive assembly



Primitive operations

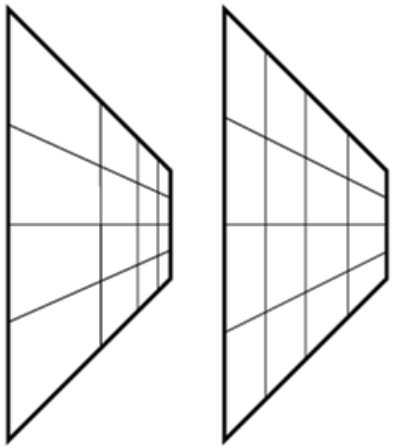


Rasterization



Perspective foreshortening

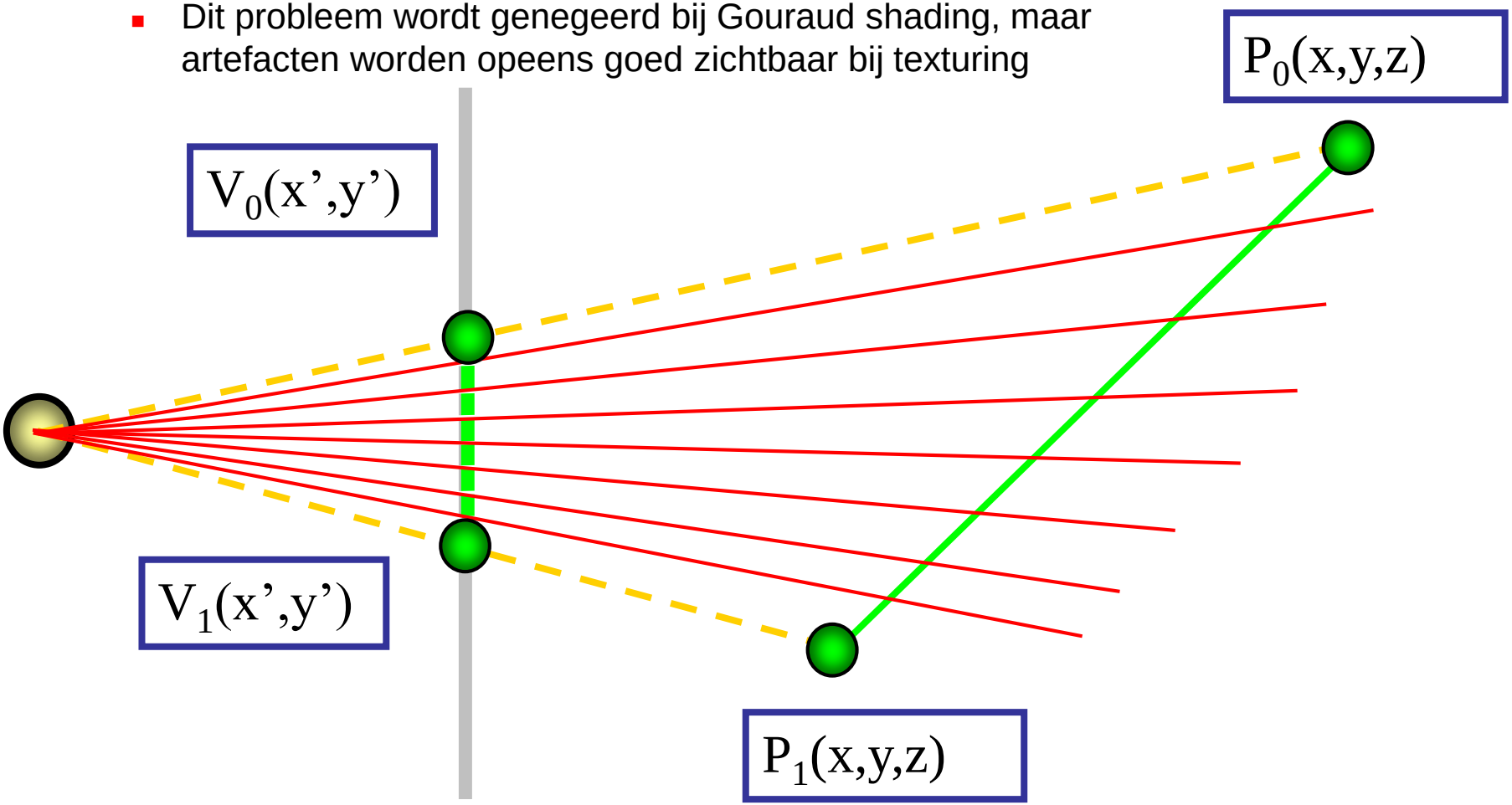
- Objecten worden kleiner naarmate de afstand tot de kijker groter wordt
- Dat willen we ook met textures maar met interpolatie over scherm(“window”)coordinaten gaat dat niet!



- Links: correct perspectief
- Rechts: interpolatie in scherm coordinaten

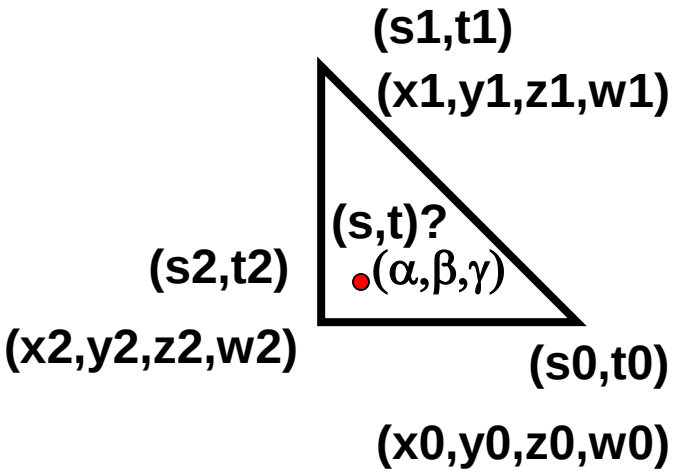
Interpolatie: scherm- vs. wereldcoordinaten

- schermcoordinaten interpolatie incorrect
  - Dit probleem wordt genegeerd bij Gouraud shading, maar artefacten worden opeens goed zichtbaar bij texturing



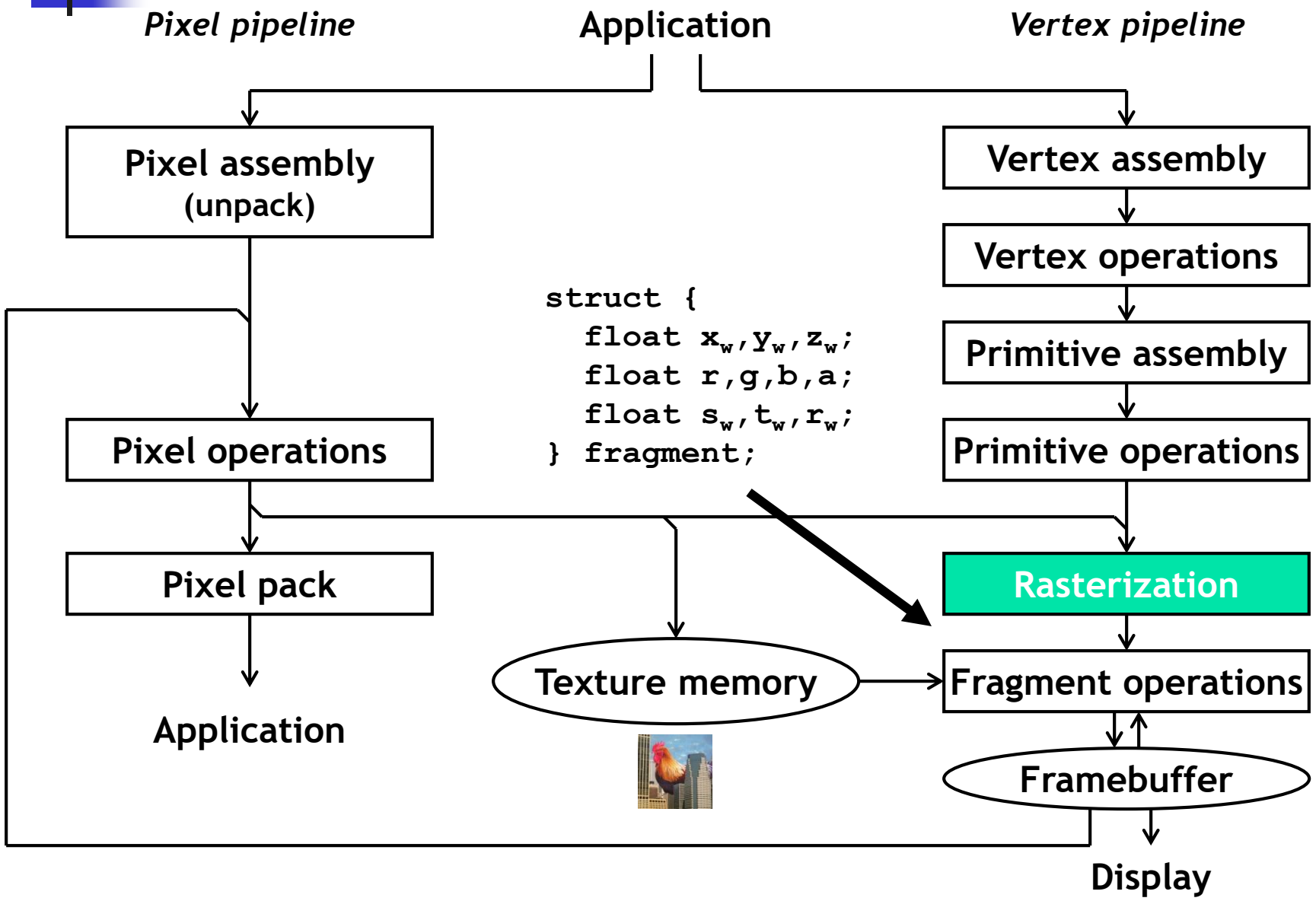
Texture Coordinaat Interpolatie

- Interpolatie met correcte perspectief: “perspective division”
  - Bereken α, β, γ :
    - barycentrische coordinaten van een punt **P** in een driehoek
    - **a** + β (**b** – **a**) + γ (**c** – **a**)
  - *s0*, *s1*, *s2* :
    - texture coordinaten van de punten
  - *w0*, *w1*,*w2* :
    - homogene coordinaten van de punten

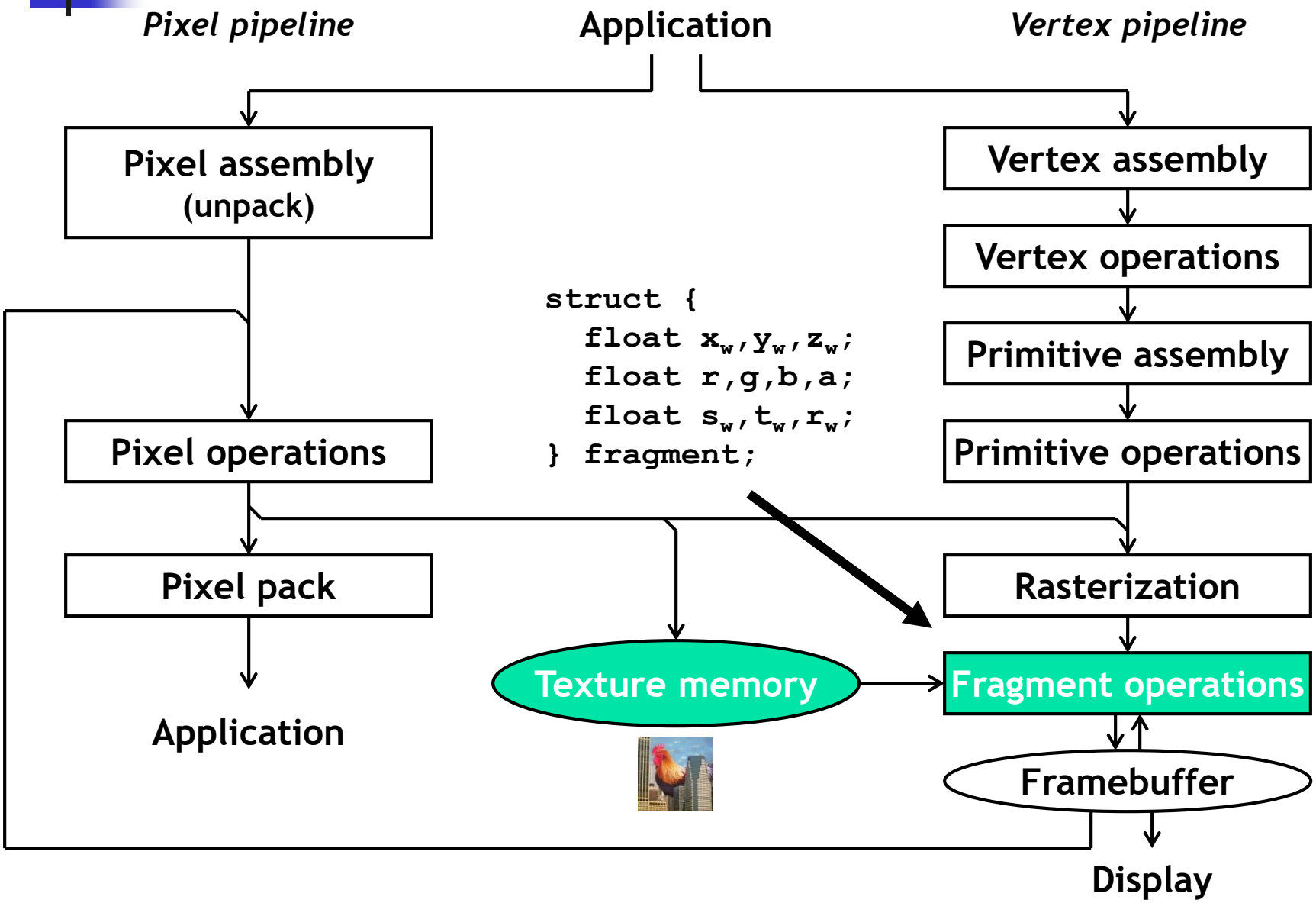


$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

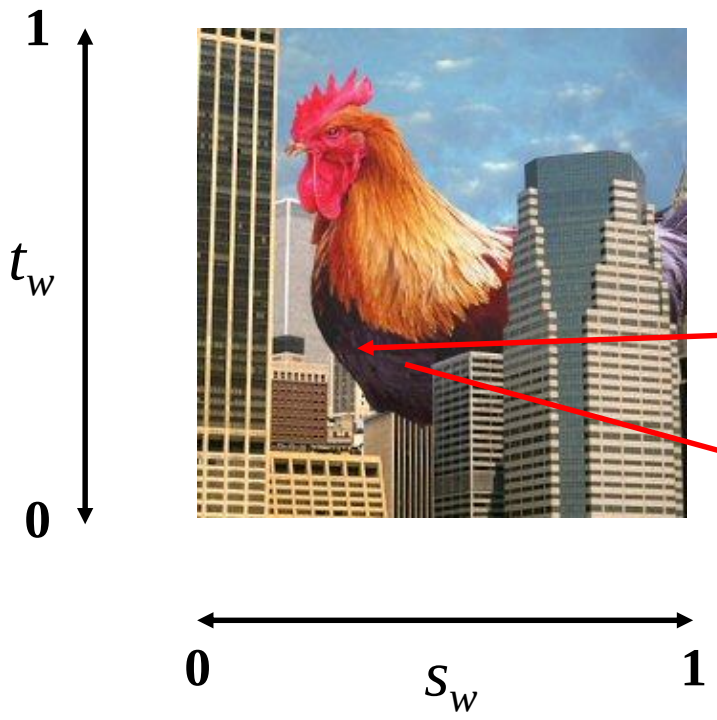
Rasterization



Fragment operations



Texture lookup




```
struct {
  float xw,yw,zw;
  float r,g,b,a;
  float sw,tw,rw;
} fragment;
```

```
struct {
  float rt,gt,bt,at;
} color;
```

If s_w or t_w is outside the range [0, 1]:

- Clamp to edge color, or
- Clamp to border color, or
- Wrap repeatedly, or
- Wrap with mirror reflections

Texture application

t_w

1

0

0

s_w

```
struct {
  float x_w,y_w,z_w;
  float r,g,b,a;
  float s_w,t_w,r_w;
} fragment;
```

```
{
t r_t,g_t,b_t,a_t;
r;
```

Alternatives include:

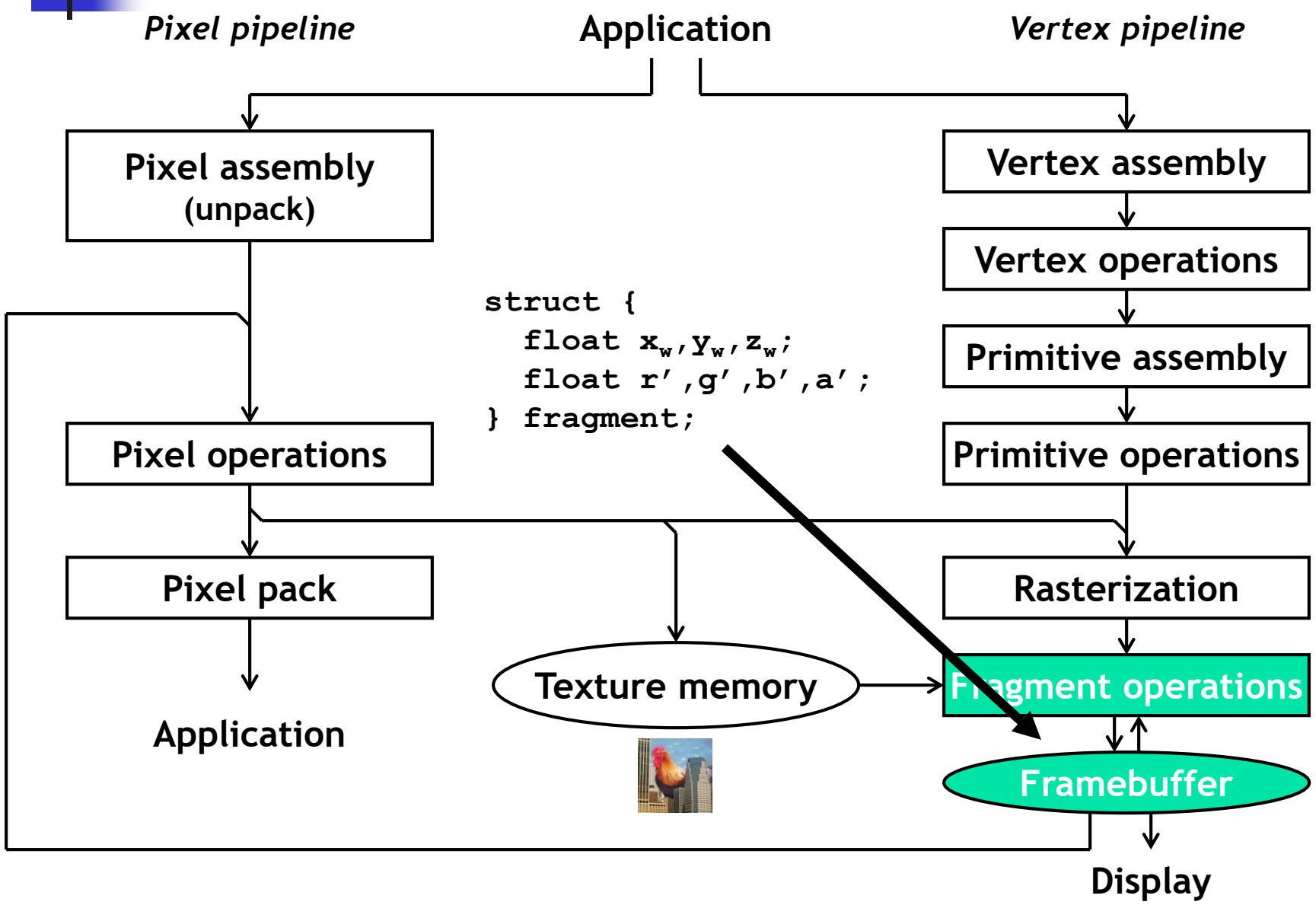
- **GL_REPLACE**
- **GL_BLEND**
- **GL_DECAL**

GL_MODULATE:

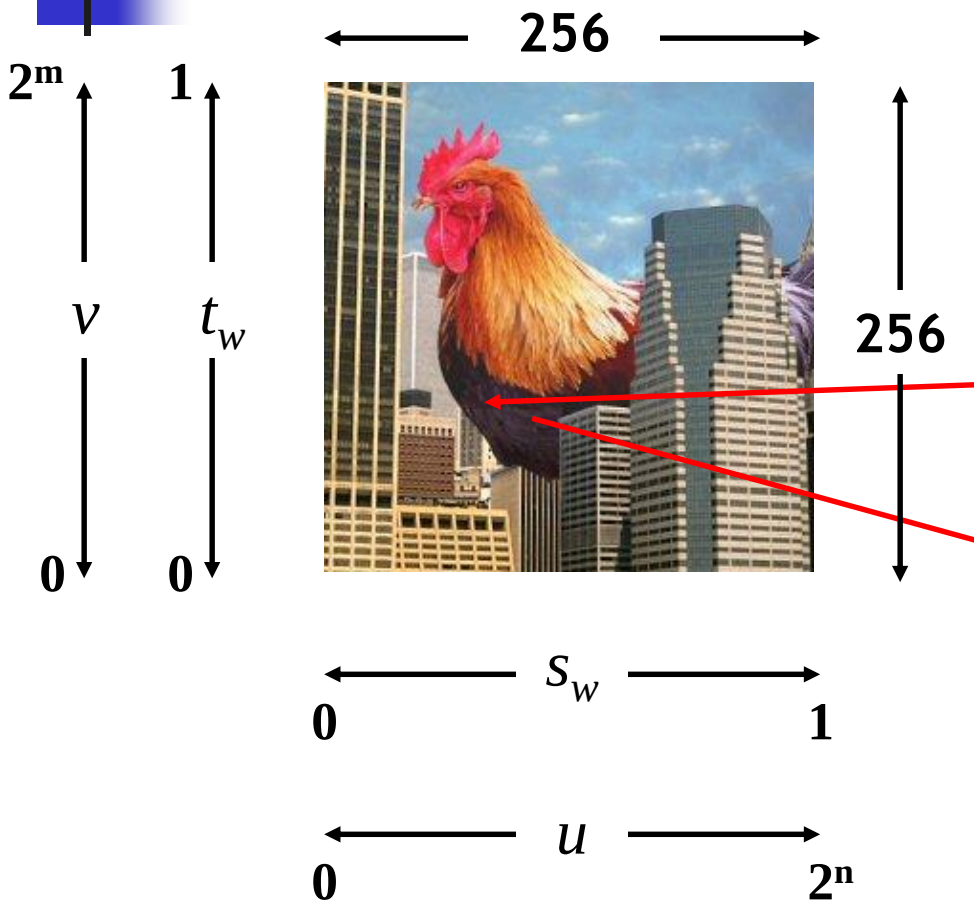
$$\begin{pmatrix} r' \\ g' \\ b' \\ a' \end{pmatrix} = \begin{pmatrix} r \cdot r_t \\ g \cdot g_t \\ b \cdot b_t \\ a \cdot a_t \end{pmatrix}$$

```
struct {
  float x_w,y_w,z_w;
  float r',g',b',a';
} fragment;
```

Fragment / framebuffer operations



Texel coordinates

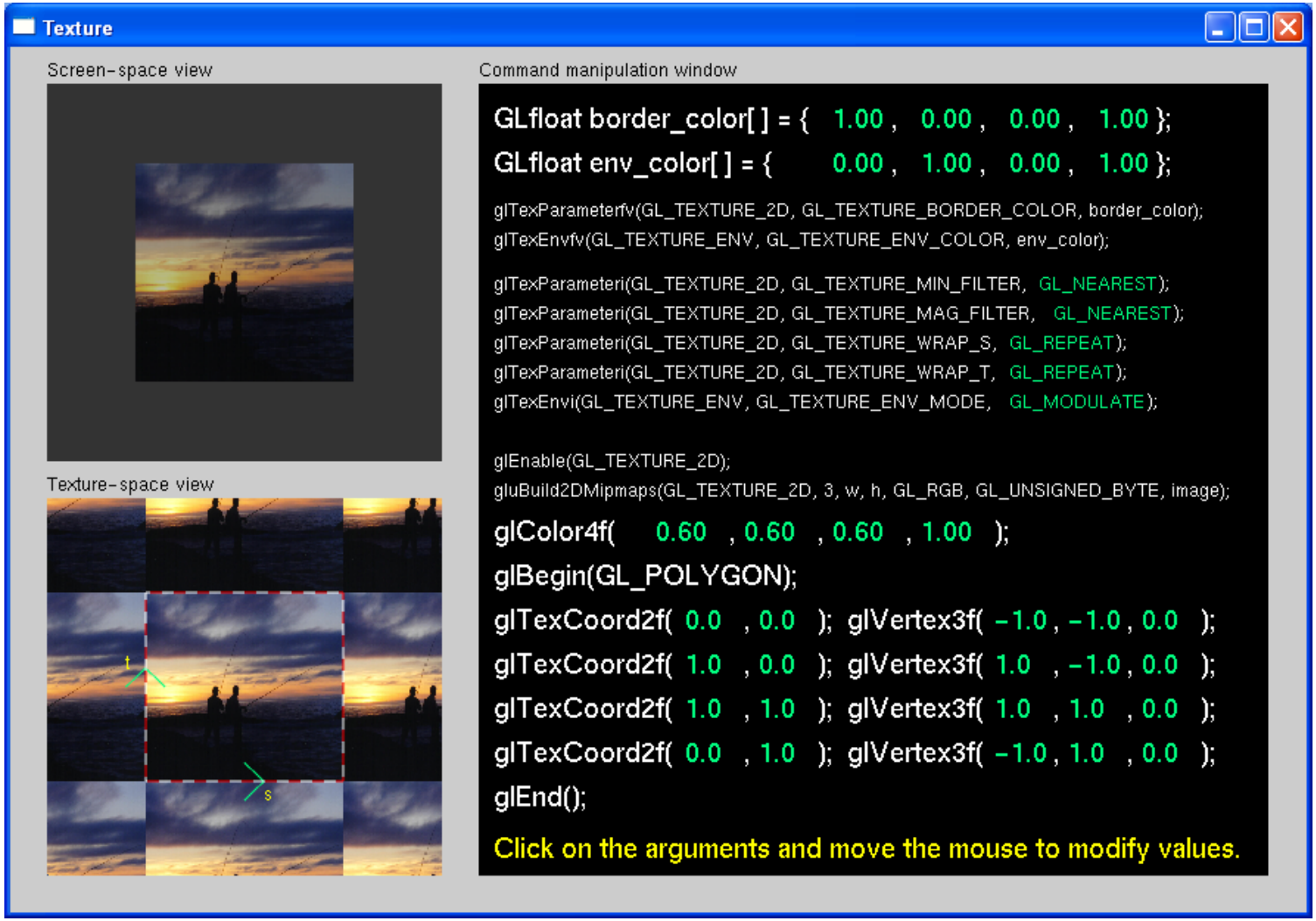



```
struct {
  float xw,yw,zw;
  float r,g,b,a;
  float sw,tw,rw;
} fragment;
```

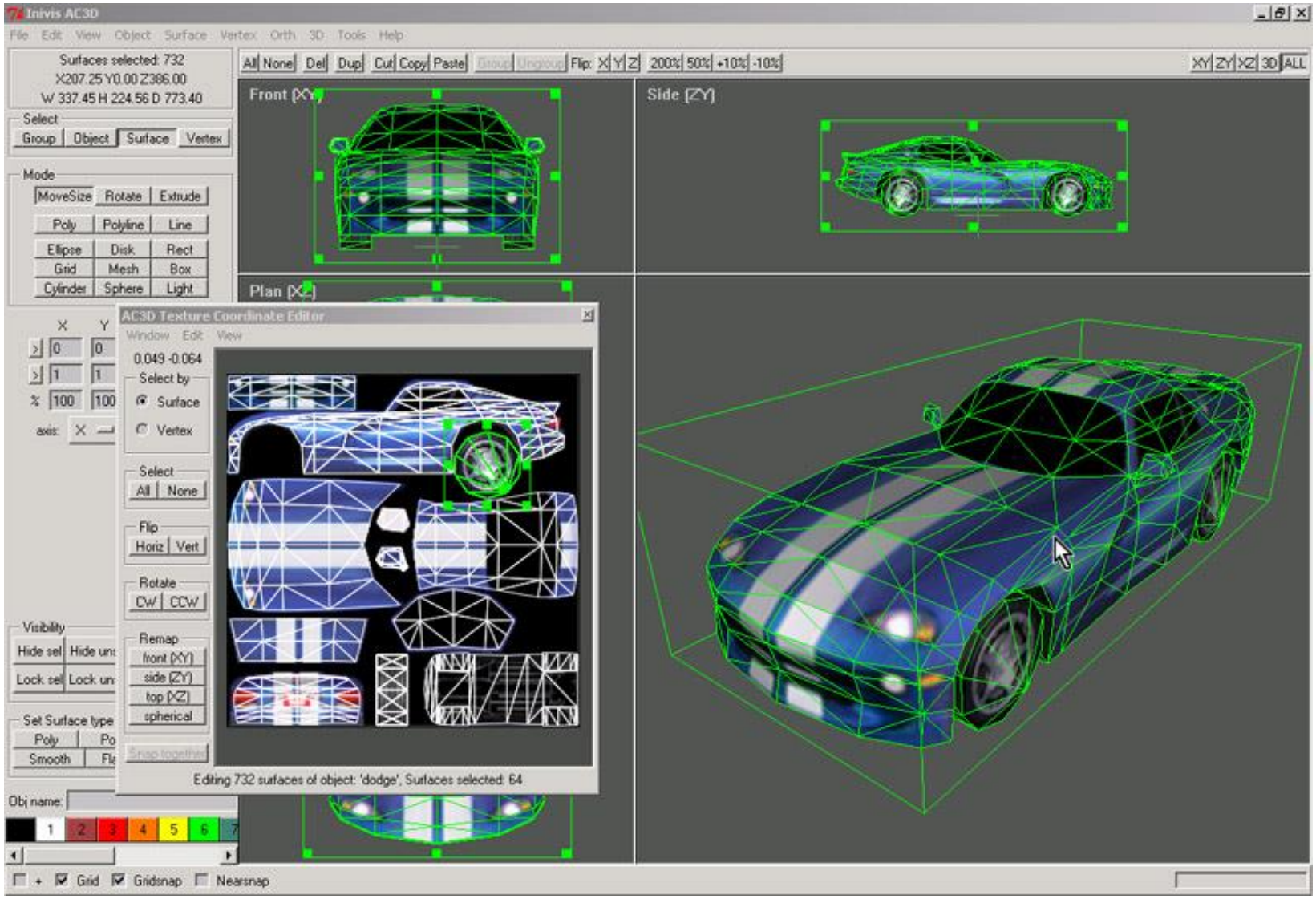
```
struct {
  float rt,gt,bt,at;
} color;
```

For this image n = m = 8

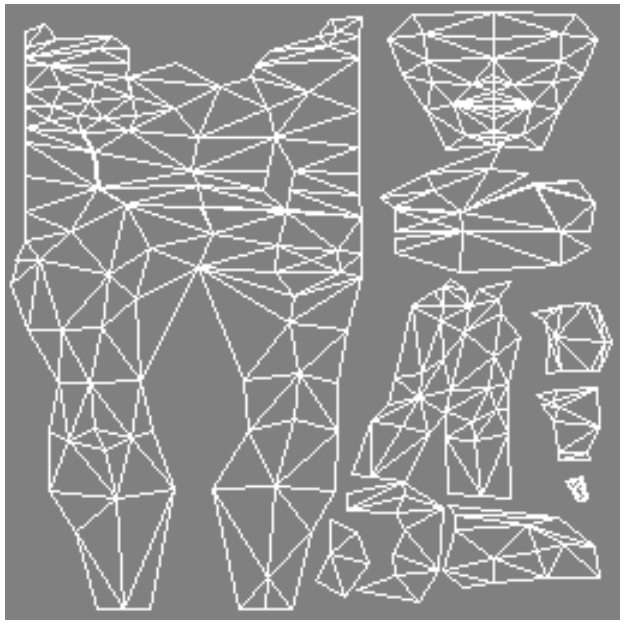
GLtutor Texture demo



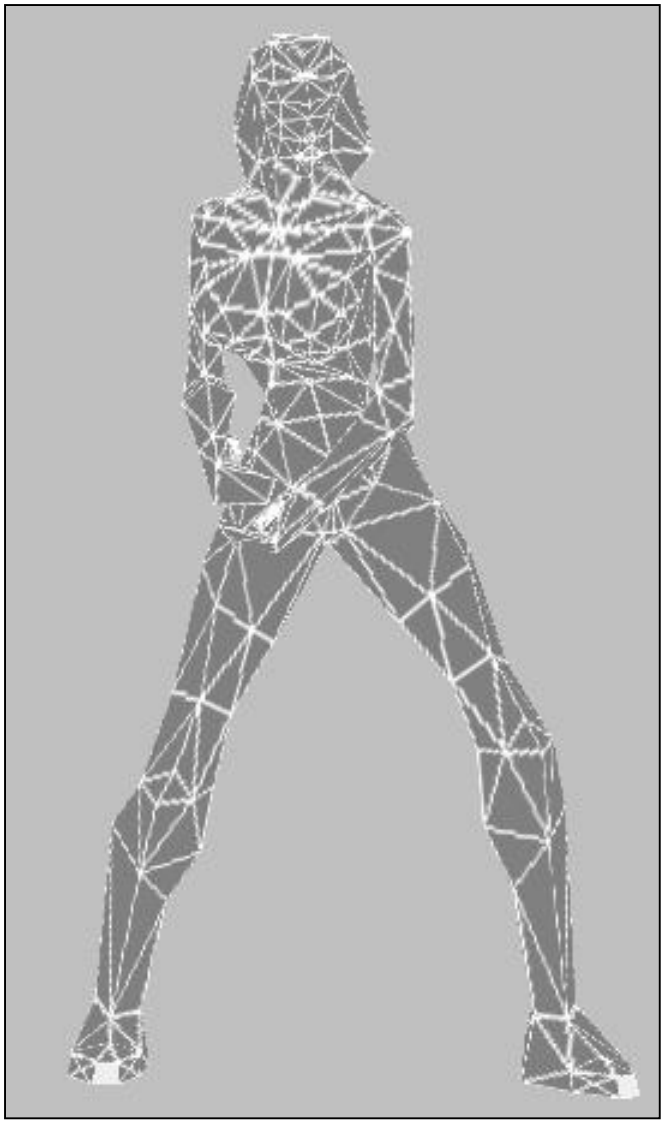
Texture coordinate editor



Example of texture maps



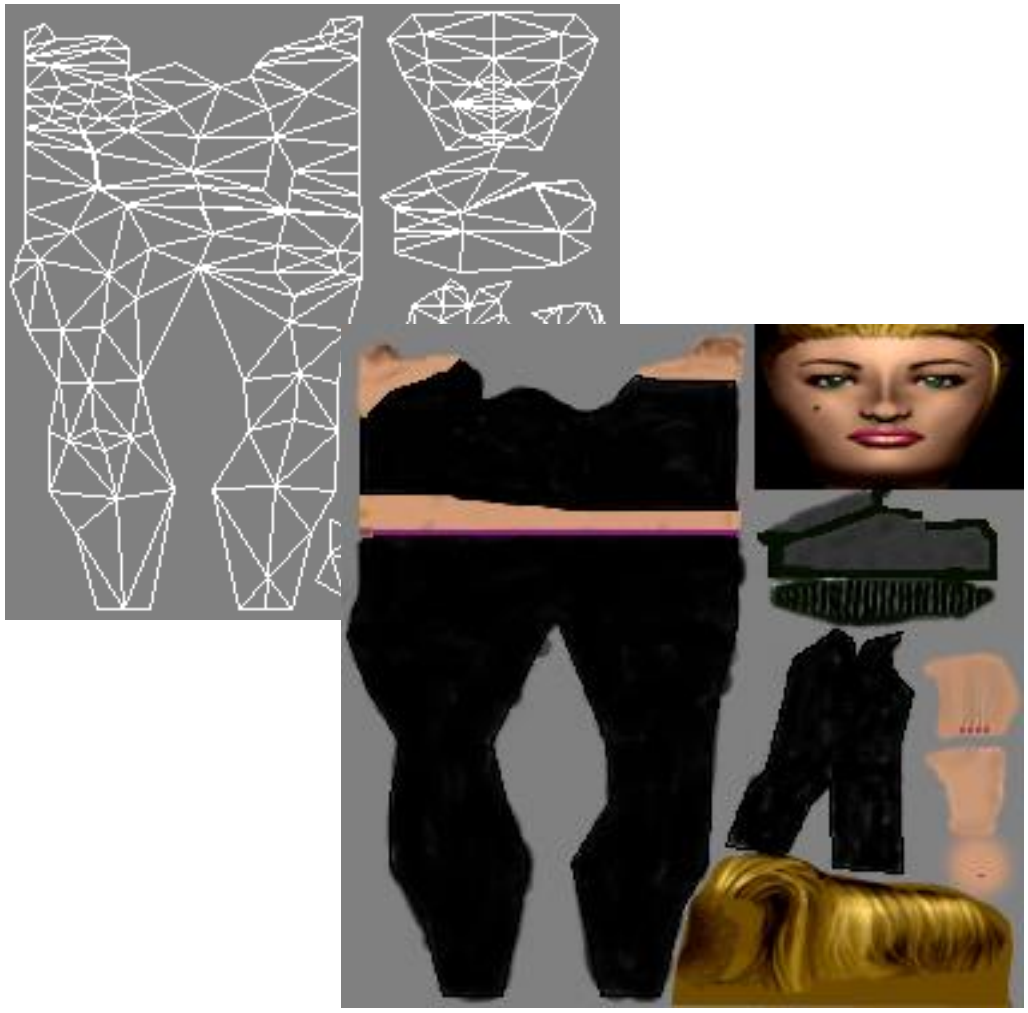
domains of the 3D model



Parametric surfaces come with a 2D domain.

Meshes: flatten parts to create a 2D domain

Example of texture maps



Sphere Mapping

- Polaire coordinaten voor een bol:

$$x = x_c + R\cos(\Phi)\sin(\Theta)$$
$$y = y_c + R\sin(\Phi)\sin(\Theta)$$
$$z = z_c + R\cos(\Phi)$$

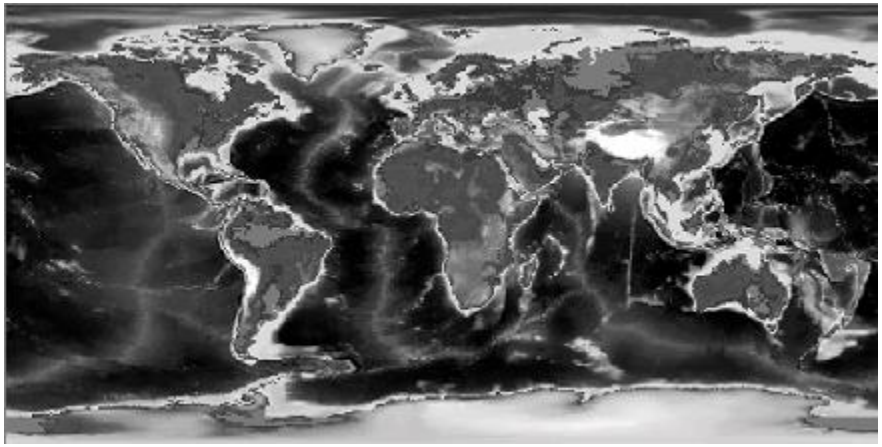
- Dan zijn Φ en Θ:

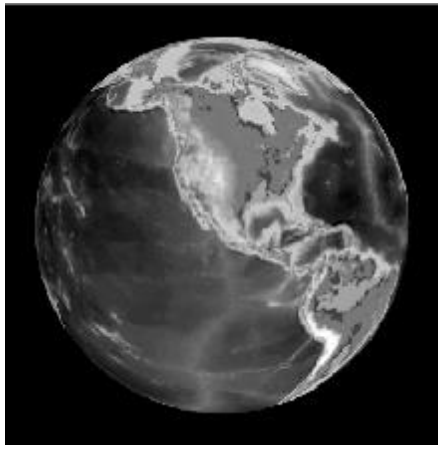
$$\Phi = \arccos\left(\frac{z - z_c}{R}\right)$$
$$\Theta = \arctan(y - y_c, x - x_c)$$

- (Φ, Θ) in interval [0,π] x [- π, π]

$$u = \frac{\Phi}{2\pi}$$
$$v = \frac{\pi - \Theta}{\pi}$$

Miller projectie





Texture Lookup: Tiling and Clamping

- Wat gebeurt er als *s* of *t* buiten het interval [0 ... 1] vallen?
- De volgende opties:
  - Negeer het deel voor de comma
    - Resulteert in een cyclische herhaling van het plaatje

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

  - Beperk de waarde tot het interval [0…1]
    - Hergebruik de kleurwaarden van de rand van het plaatje

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

- [GLtutor texture demo]

Fractional Texture Coordinates

texture image

(0,1) **(1,1)**

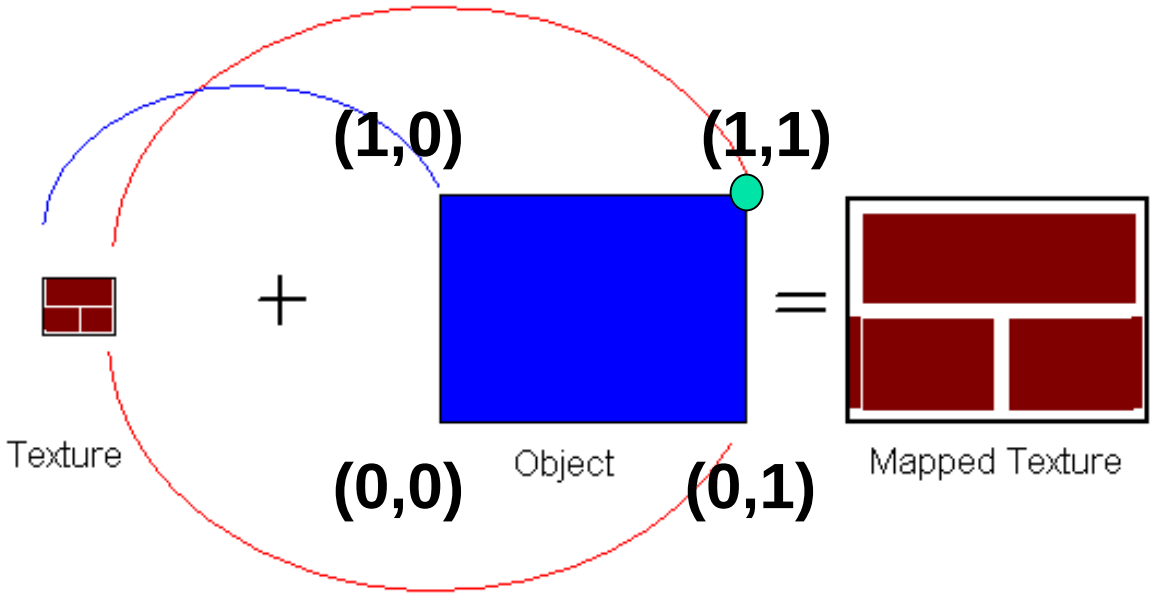
(0,0) **(1,0)**

(0,.5) **(.25,.5)**

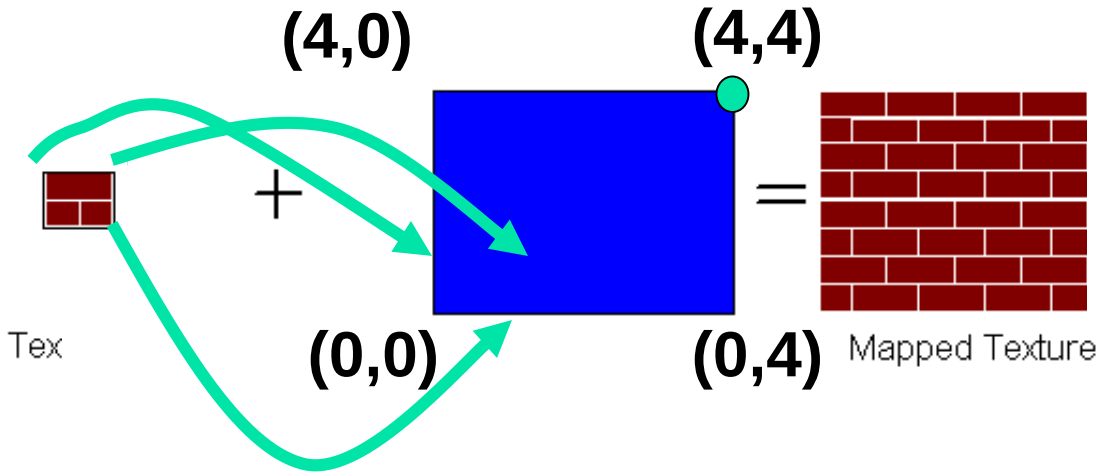
(0,0) **(.25,0)**

Tiled Texture Map

glTexCoord2d(1, 1);
glVertex3d (x, y, z);



glTexCoord2d(4, 4);
glVertex3d (x, y, z);

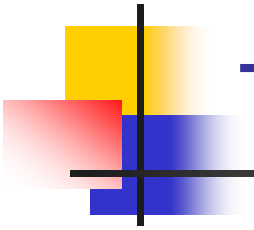


Texture coordinaat transformaties

- Waarom?
  - Verander schaal en oriëntatie van een texture op een object
- Hoe?
  - Naast **`GL_MODELVIEW`** en **`GL_PROJECTION`** hebben we hiervoor nog een matrix stack: **`GL_TEXTURE`**
  - Transformaties toegepast op texture coorindates:

```
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glRotate();
...
```

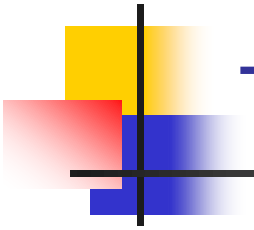
  - Biedt meer flexibiliteit dan alleen (s,t) coordinaten
- [GLtutor texture demo]



Texture “modes”

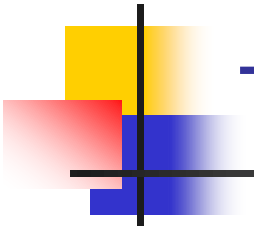
- Hoe wordt de waarde uit het texture toegepast op het oppervlak? De volgende opties:
  - Direct toepassen als oppervlaktekleur: **GL_REPLACE**
    - Gooi oude kleur weg
      - Belichtingsberekening ongebruikt
  - Moduleer met de huidige oppervlaktekleur: **GL_MODULATE**
    - Vermenigvuldig de huidige oppervlaktekleur met de texture kleur
      - Behoudt belichtingsberekening
      - Gebeurt *na* belichting, niet opnieuw belicht
  - Toepassen als oppervlaktekleur, moduleer alpha: **GL_DECAL**
    - Als `GL_REPLACE`, maar met ondersteuning voor texture transparency
  - Meng oppervlaktekleur met nieuwe kleur: **GL_BLEND**
    - Texture kleur bepaalt welke van de twee kleuren te gebruiken
    - Indirect; de texture kleur niet onmiddelijk voor kleuring gebruikt

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <mode>)
```



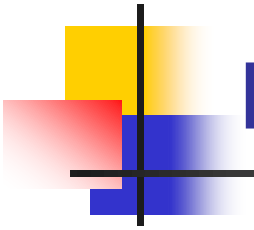
Texture objects en texture binding

- Texture object:
  - Een data type dat textures in geheugen houdt
    - Met identifiers om ze te benaderen
  - Efficiënter dan herhaaldelijk textures inladen
  - Texture objecten kunnen geprioritiseerd worden
    - OpenGL gebruikt “least recently used” (LRU) als geen prioriteit is toegewezen
- Texture binding:
  - Bepaal welk texture moet worden gebruikt
    - Aan de hand van de identifiers
  - Selecteer tussen ingeladen textures



Texturing in OpenGL

- Maak een texture object vul het met data:
  - Maak een texture object: levert identifier
    - `glGenTextures(num, &identifiers)`
  - Bind texture commandos aan een texture object
    - `glBindTexture(GL_TEXTURE_2D, identifiers[i])`
  - Specificeer hoe texture gesampled moet worden
    - `glTexParameteri(GL_TEXTURE_2D, …, …)`
  - Specificeer texture image (het plaatje)
    - `glTexImage2D(GL_TEXTURE_2D, …)`
- Zet texturing aan:
    - `glEnable(GL_TEXTURE_2D)`
- Specificeer tiling/clamping van texture:
    - `glTexEnvf(…)`
- Specificeer texture coordinaten voor elke punt op oppervlak:
    - `glTexCoord2f(0,0); glVertex3f(x,y,z);`



Details: `glTexImage2D`

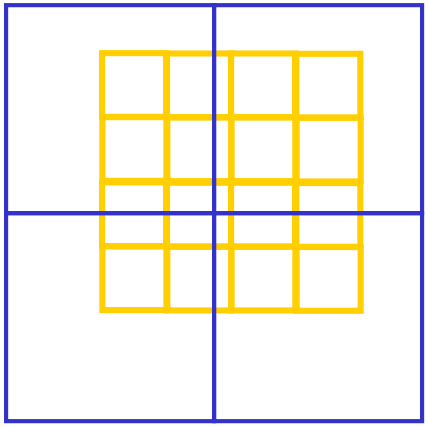
- Voor het specificeren van een texture bestaat een berg functies om aan te geven:
  - wat de layout van een plaatje is,
  - aantal bits per pixel,
  - welke kanalen er in zitten (RGB, RGBA, LUMINANCE, etc.)

```
void glTexImage2D( GLenum target,
                   GLint level,
                   GLint internalFormat,
                   GLsizei width,
                   GLsizei height,
                   GLint border,
                   GLenum format,
                   GLenum type,
                   const GLvoid *pixels )
```

- Voor OpenGL versie 2.0 moesten de breedte en hoogte van een texture een 2-macht zijn
  - Kleiner neemt minder geheugen in
  - Texture memory zit op de grafische kaart en die is vaak beperkt

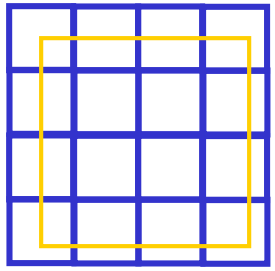
Reconstructie

- Hoe ga je om met:
  - pixels die veel kleiner zijn dan een texel?
    - filtering, “averaging”: nearest neighbour, linear interpolation



```
glTexParameteri(GL_TEXTURE_2D,
   GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

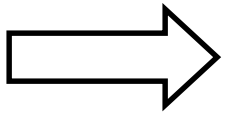
  - pixels die veel groter zijn dan een texel?
    - interpolatie: nearest neighbour, linear interpolation, nearest mipmap nearest neighbour nearest mipmap linear, linear mipmap nearest neighbour, linear mipmap linear



```
glTexParameteri(GL_TEXTURE_2D,
   GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

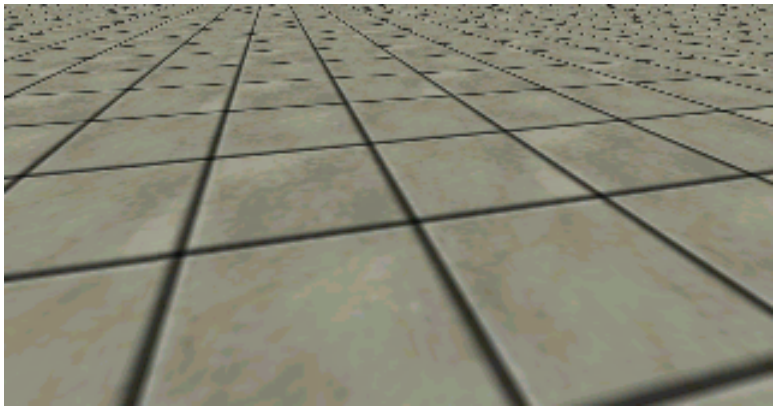
Nog een keer: textured quad

```
glGenTextures(1, &i);          // create 1 texture object
LoadTexture("rooster", i);     // load file into texture object
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, i);
glClearColor(1, 1, 1, 1);      // white
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glOrtho(0, 100, 0, 100, -1, 1);
glColor3f(1, 1, 1);            // white
glBegin(GL_TRIANGLE_STRIP);
  glTexCoord2f(0, 0);          glVertex2i(11, 31);
  glTexCoord2f(0, 1);          glVertex2i(37, 71);
  glTexCoord2f(1, 0);          glVertex2i(91, 38);
  glTexCoord2f(1, 1);          glVertex2i(65, 71);
glEnd();
glFlush();
```

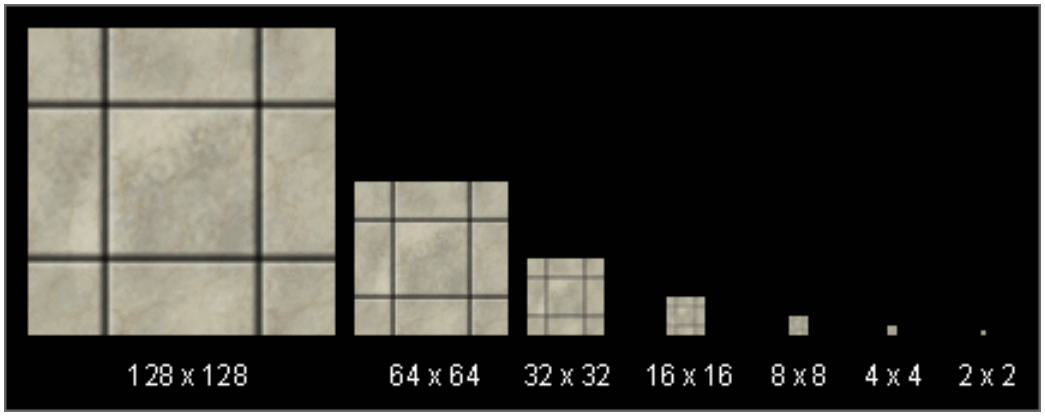


MIPmapping

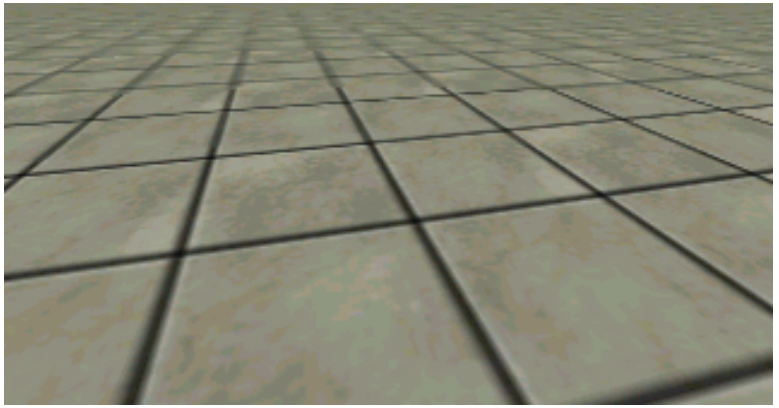
- Gebruikt een “image pyramid” met op voorhand berekende verkleinde en gefilterde versies van de texture
- De hele “image pyramid” zit in een blok geheugen



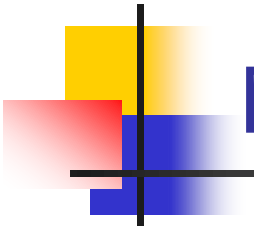
Texturing zonder MIP-mapping



MIPmap



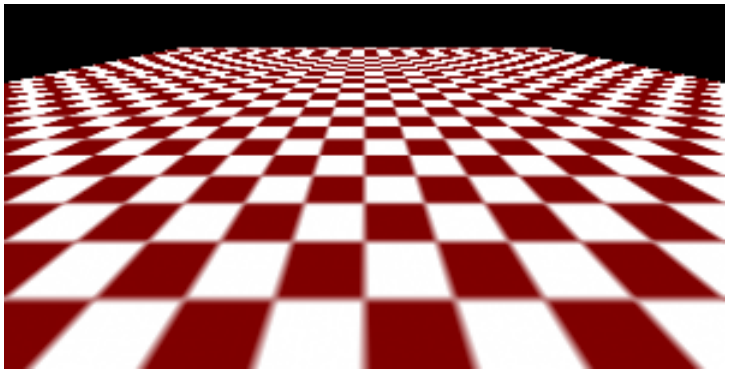
Texturing met MIP-mapping



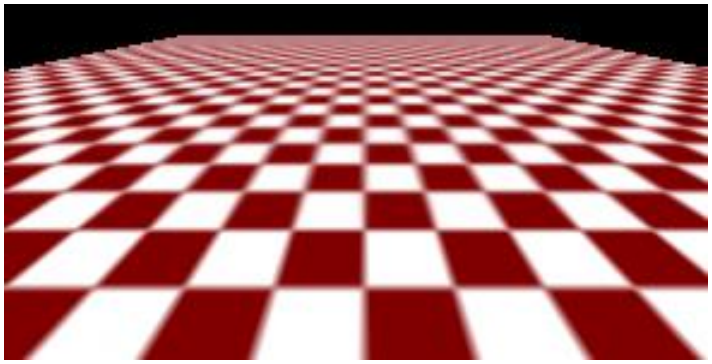
MIPmaps

- **multum in parvo** -- many things in a small place
  - Specificeer een serie van texture maps van afnemende resolutie
  - Meer geheugenruimte nodig (slechts 1/3 meer; zie volgende sheet)
  - Voorkomt flikkeren als objecten bewegen
- **`gluBuild2DMipmaps`** functie
  - Maakt automatisch een serie textures van de oorspronkelijke texture grootte tot 1x1

zonder

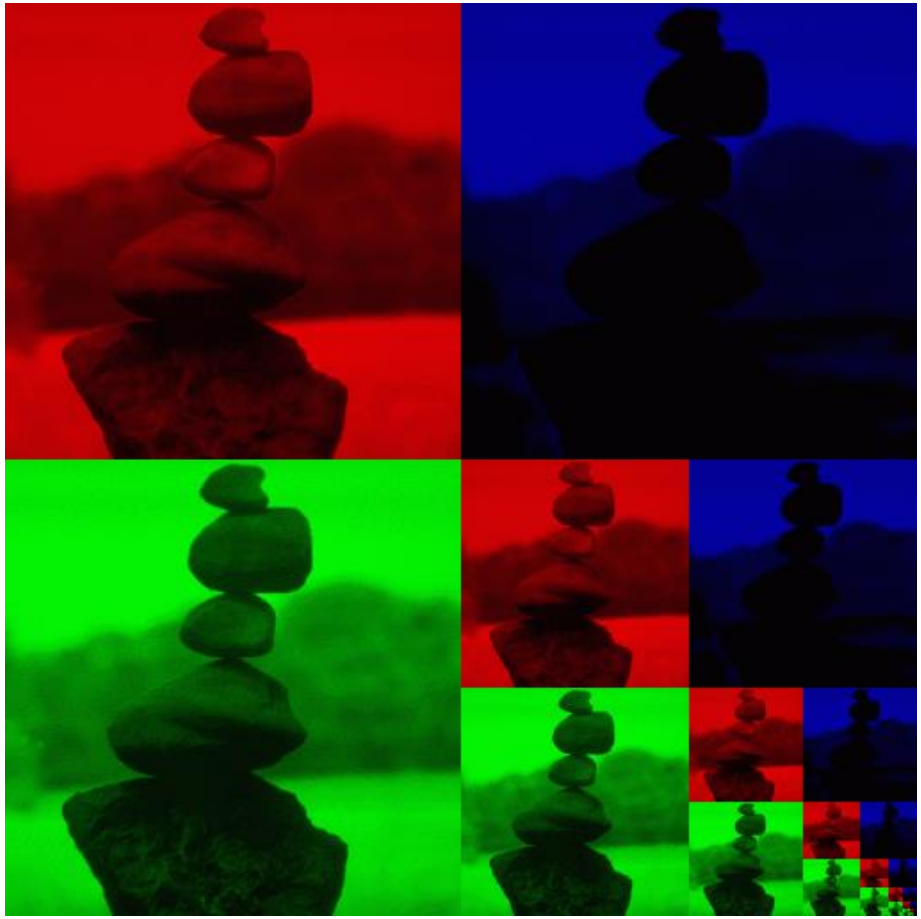


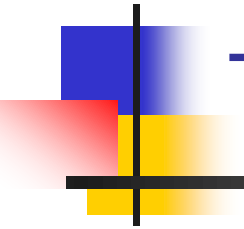
met



MIPmap opslag

- Slechts 1/3 meer ruimte nodig!



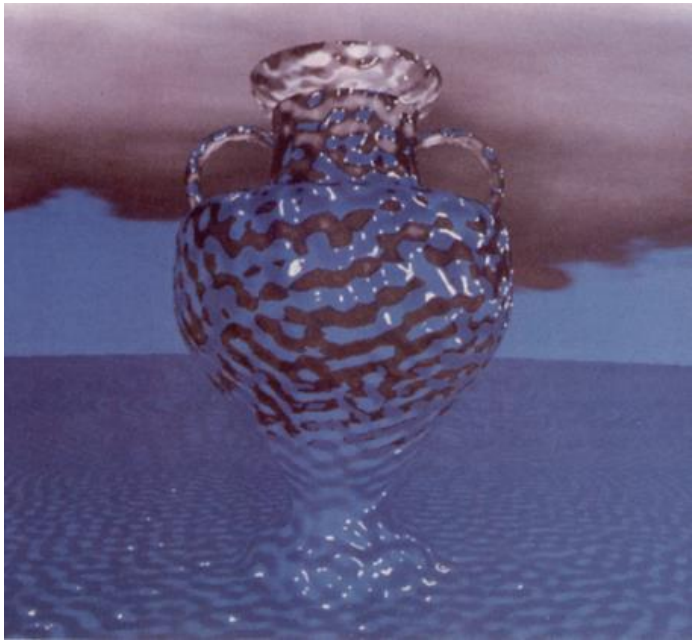


Toepassingen van texture mapping

- Bump mapping
- Displacement mapping
- Environment mapping
- Direct volume rendering

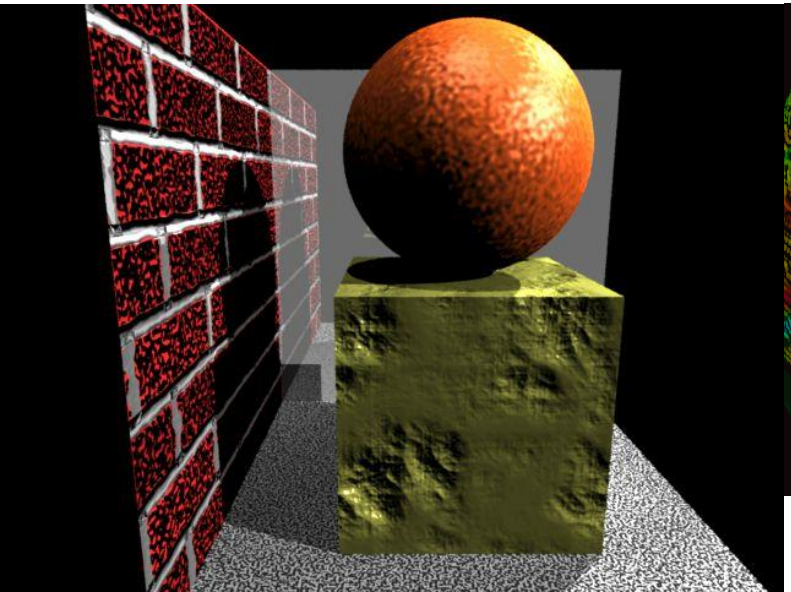
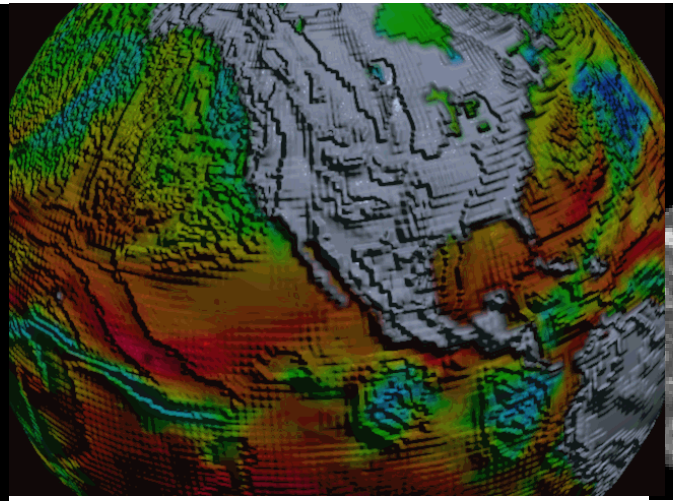
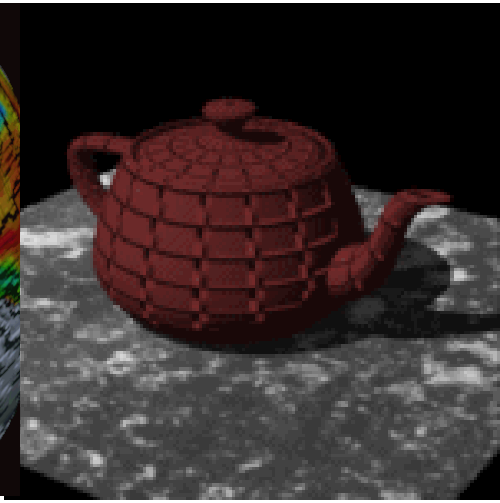
Texture Parameters

- Behalve kleur is het ook mogelijk om andere materiaal of object eigenschappen te veranderen:
  - surface normal (bump mapping)
  - reflected color (environment mapping)

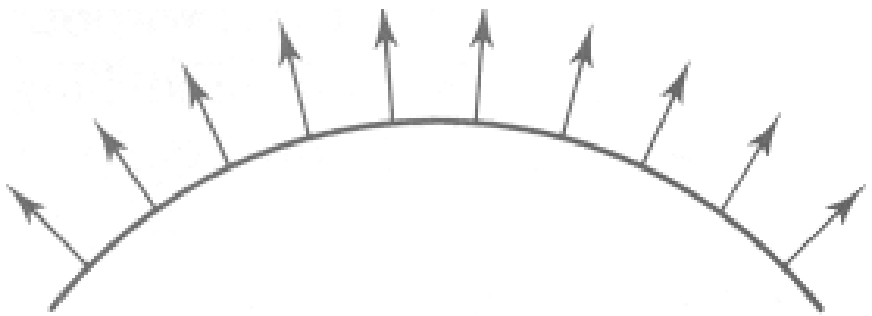


Bump Mapping

- Het oppervlak van een object is vaak “ruw”
  - Namaken in de vorm van geometrie is te complex
- Door lokaal de normaal vector te beïnvloeden kan je dit effect ook bereiken
  - random
  - over een regio

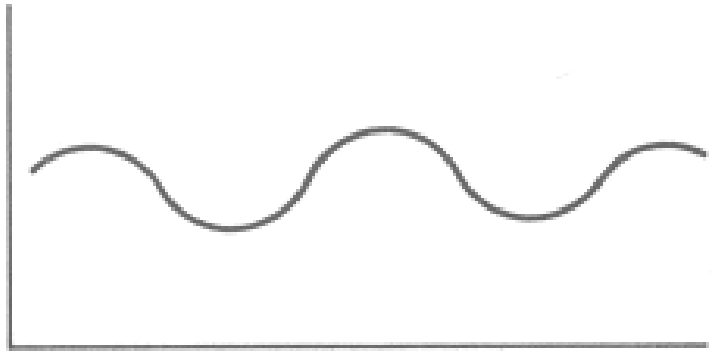




Bump Mapping



$O(u)$

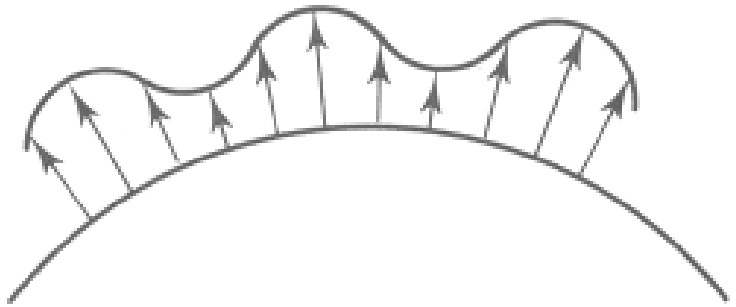
Original surface



$B(u)$

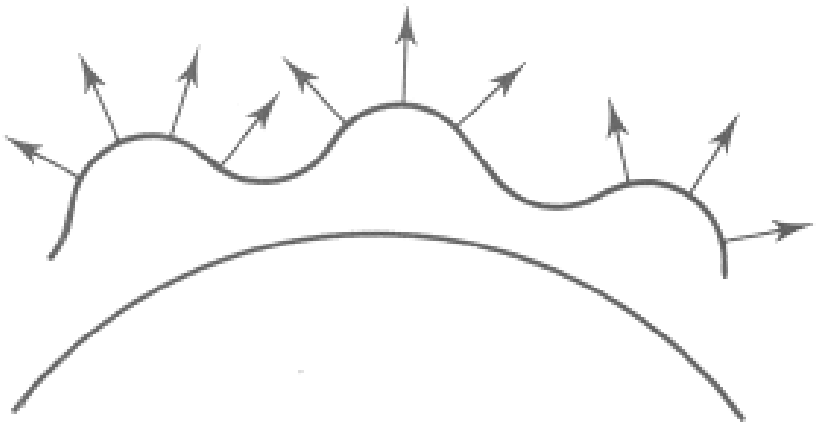
A bump map

Bump Mapping



$O'(u)$

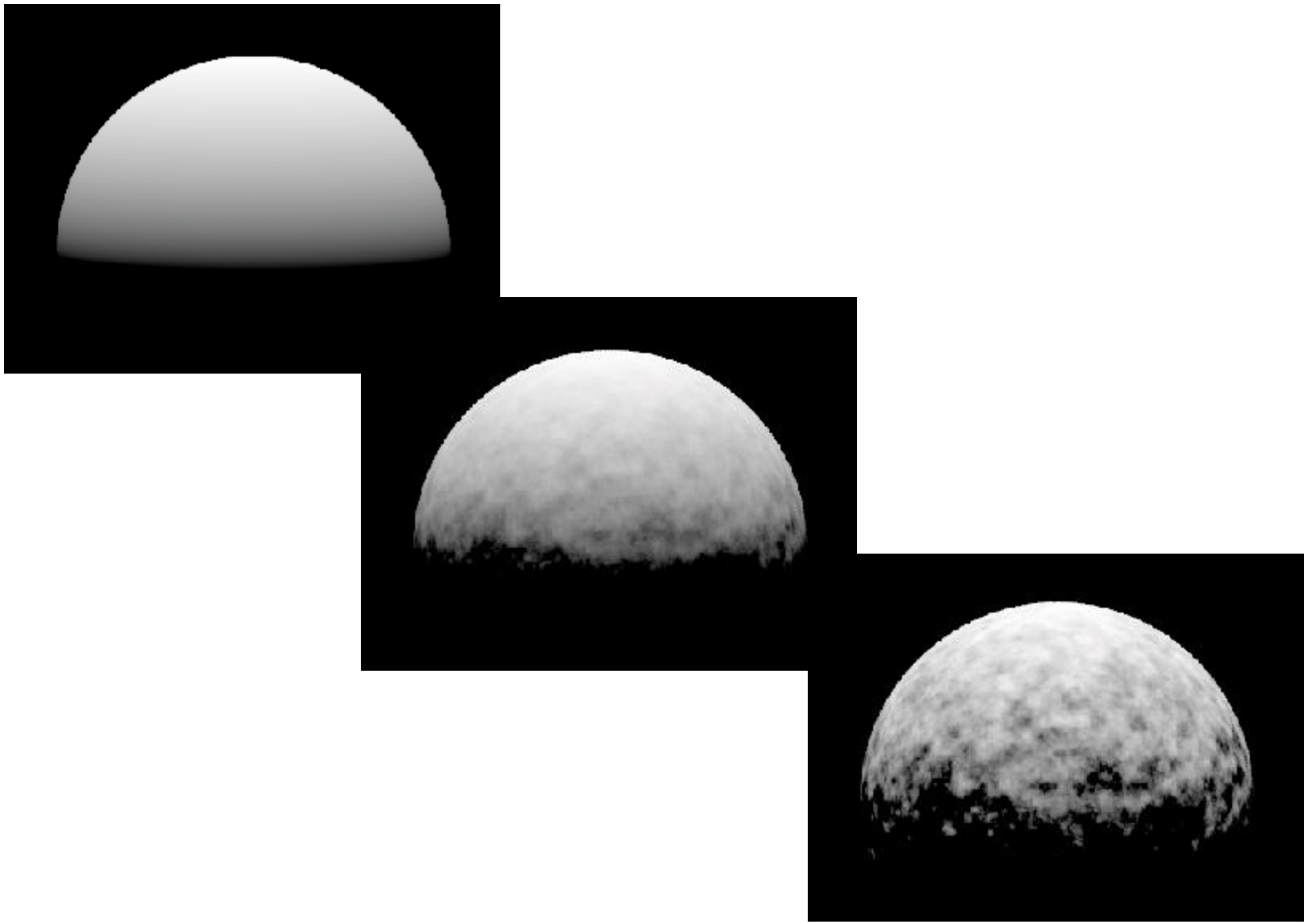
Lengthening or shortening $O(u)$ using $B(u)$



$N'(u)$

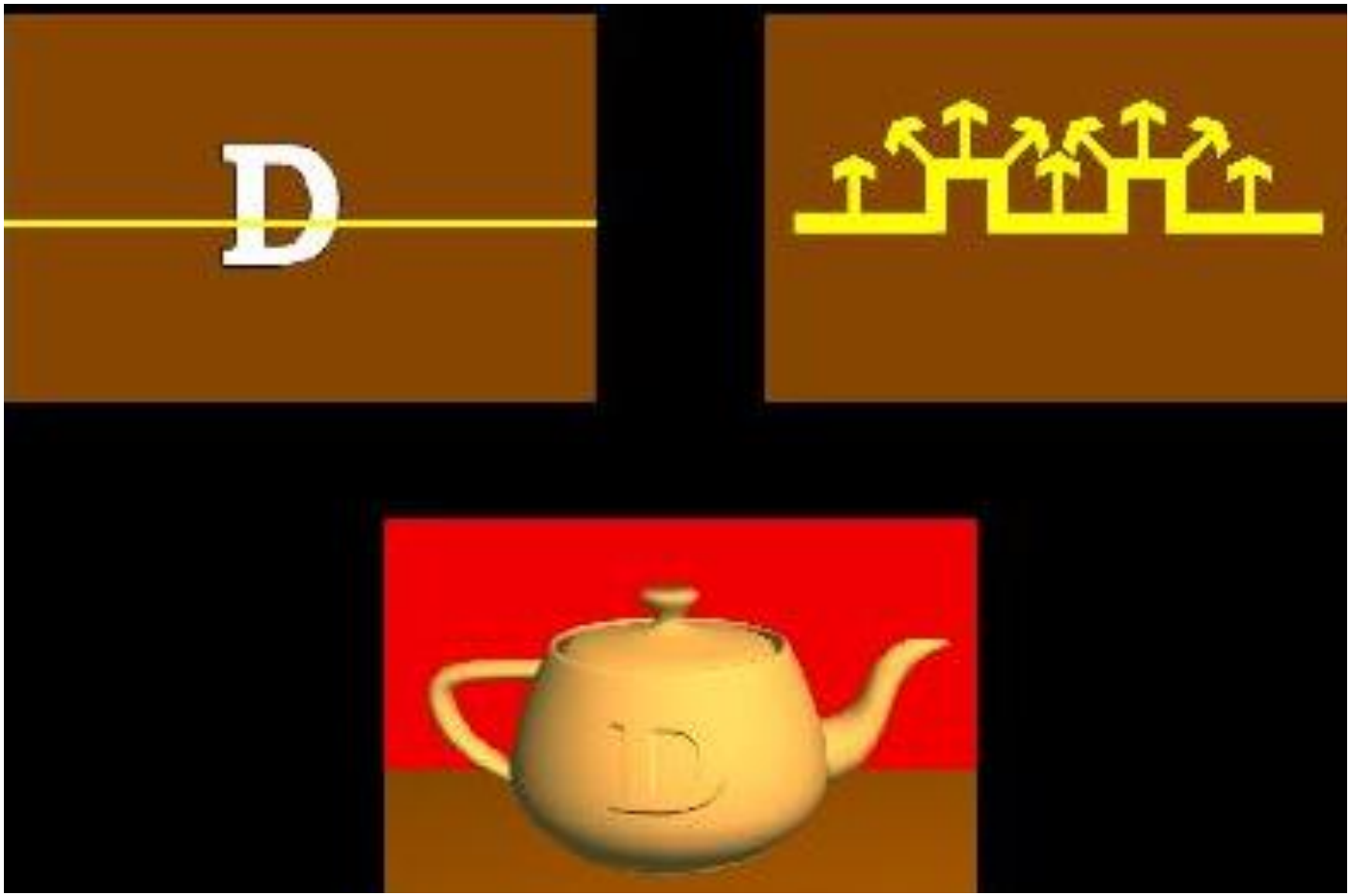
The vectors to the 'new' surface

Bump mapping



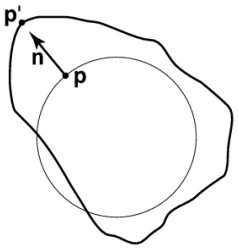
Embossing

- Bij overgangen
  - Roteer normaal van een punt met een hoek van θ or $-\theta$



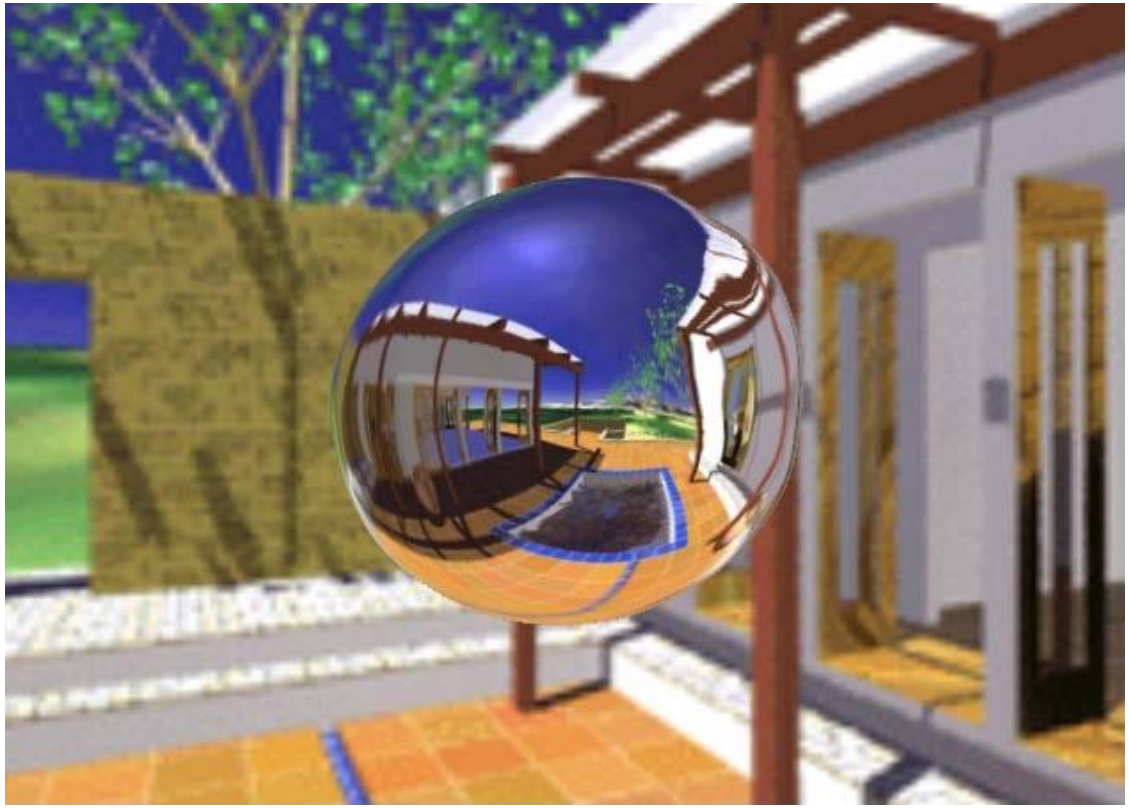
Displacement Mapping

- Bump mapping doet het niet goed als je kijkt naar het silhouette
  - Schaduwen zijn ook vaak verkeerd
- Oplossing: verander de opper-vlaktestructuur
  - Dit is vrij nieuw: “geometry shaders”
  - Moet dan wel mogelijk zijn om het oppervlak op te delen



Environment Mapping

- Goedkope manier om “reflectie effect” te maken
  - Maak een plaatje van de omgeving
  - Plak het op een object als texture



Environment Mapping

- Wordt gebruikt om een object te modeleren dat de omgeving in het oog reflecteert
  - Eerst gebruikt in cyborg in Terminator 2
- Verschillende benaderingen:
  - Bol en kubus meest populair
    - OpenGL ondersteuning:
      - `GL_SPHERE_MAP, GL_CUBE_MAP`
  - Andere zijn ook mogelijkheid

Cyborg van vloeibaar metaal in Terminator 2

Sphere Mapping

- De texture is een vervormd “fish-eye” aanzicht
  - Richt de camera naar een “spiegelbol”
  - Spherische texture mapping maakt texture coordinaten die correct indiceren in de texture map

Sphere Mapping

- Polaire coordinaten voor een bol:

$$x = x_c + R\cos(\Phi)\sin(\Theta)$$
$$y = y_c + R\sin(\Phi)\sin(\Theta)$$
$$z = z_c + R\cos(\Phi)$$

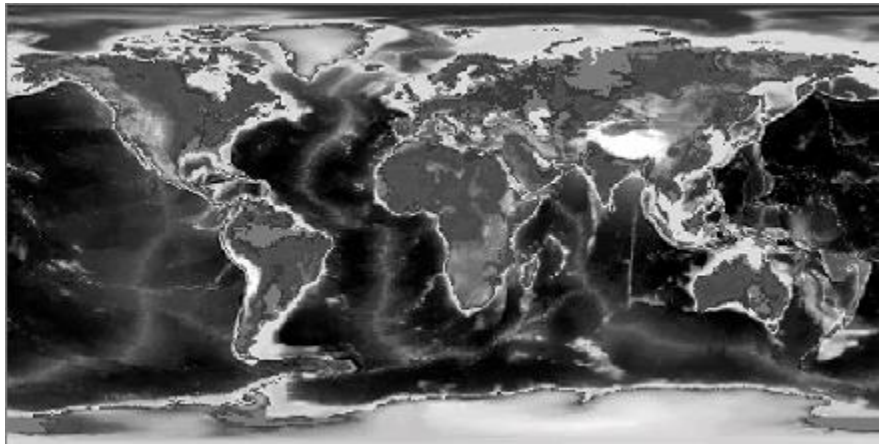
- Dan zijn Φ en Θ:

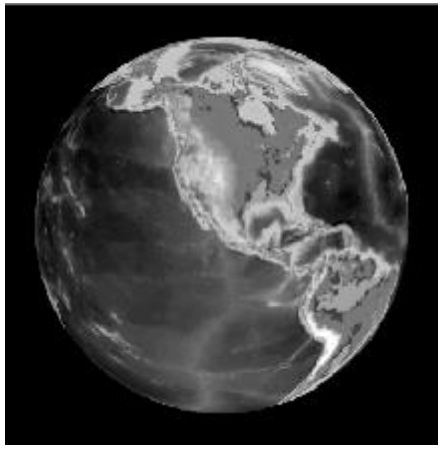
$$\Phi = \arccos\left(\frac{z - z_c}{R}\right)$$
$$\Theta = \arctan(y - y_c, x - x_c)$$

- (Φ, Θ) in interval [0,π] x [- π, π]

$$u = \frac{\Phi}{2\pi}$$
$$v = \frac{\pi - \Theta}{\pi}$$

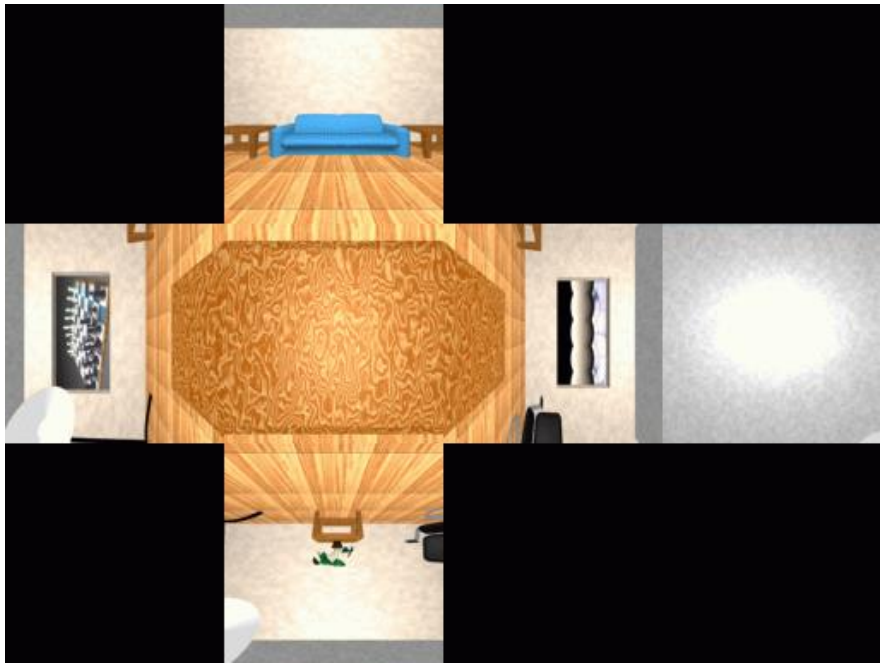
Miller projectie



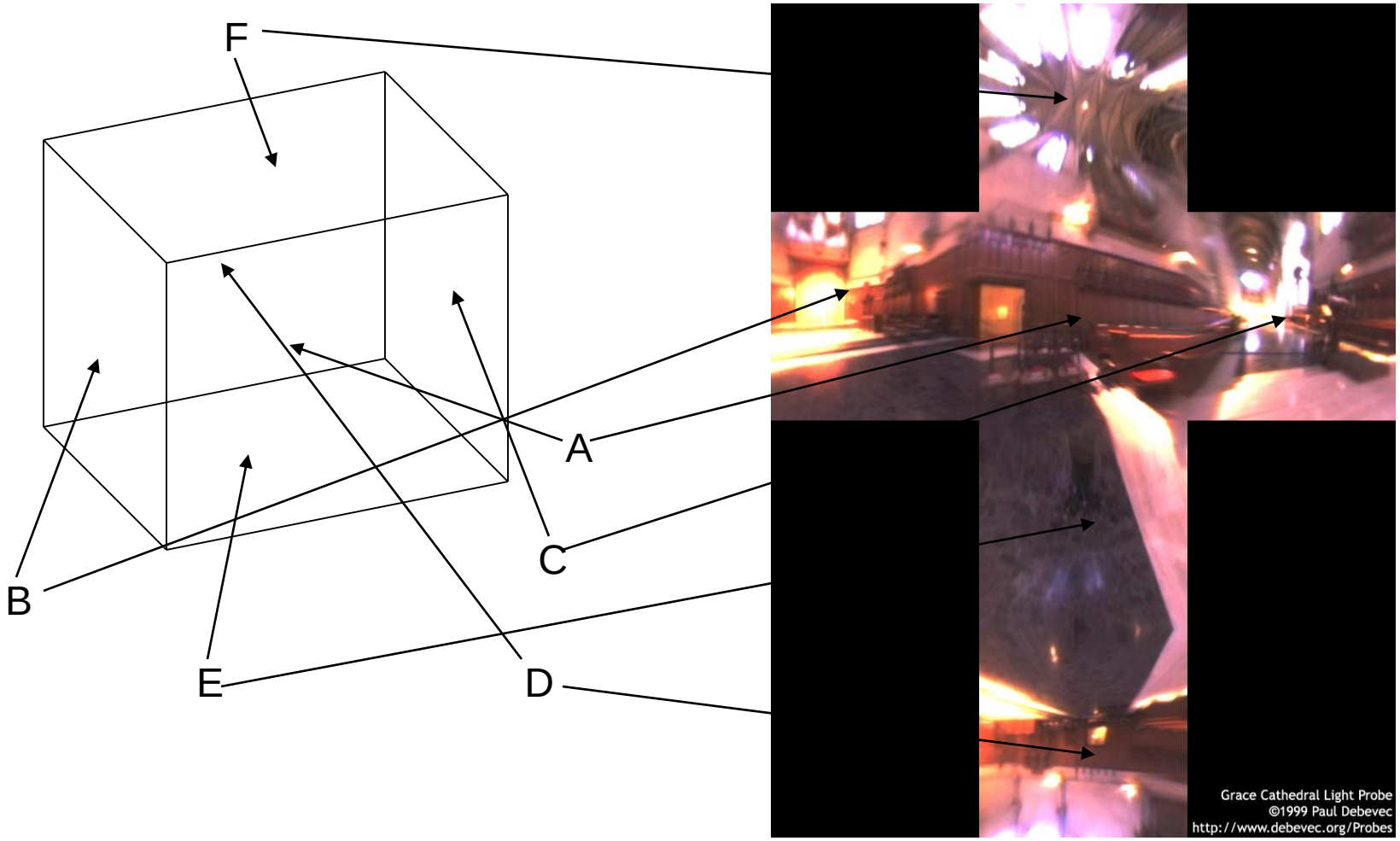


Cube Mapping

- 6 2D textures: zijden van een oneindig grote kubus
  - Richt camera in 6 verschillende richtingen, weg van de oorsprong



Cube Mapping

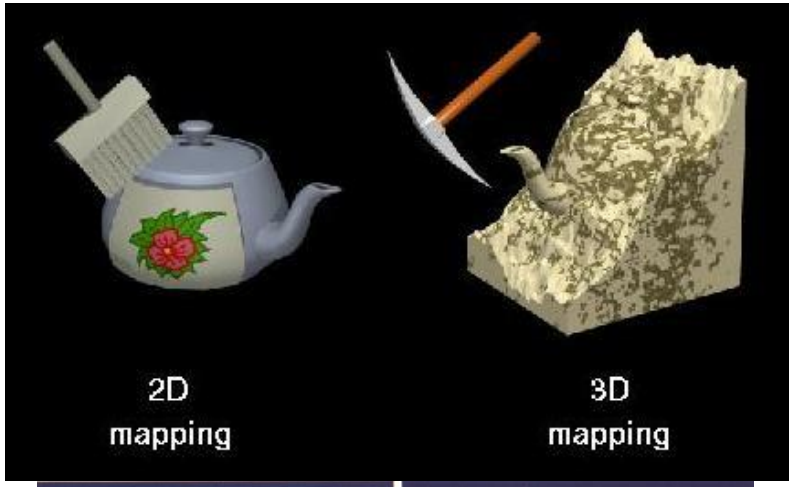


Cube Mapping

- De richting van de reflectievector ***r*** bepaalt welke zijde van de kubus wordt gebruikt
  - De coordinaat met de grootste absolute waarde
    - B.v. de vector (-0.2, 0.5, -0.84) selecteert de –Z zijde
  - De overige twee coordinaten, genormaliseerd naar de 3e coordinaat, selecteert de pixel uit de zijde
    - B.v. (-0.2, 0.5) wordt dan (0.38, 0.80)
- Moeilijk bij interpoleren over de zijden

Volumetrische Textures

- Texture gedefinieerd over 3D domein - 3D ruimte waar het object “in zit”
  - Texture functie vaak gesampled of procedureel
  - Voor elk punt op een object bepaal texture uit de lokatie van het punt in de ruimte
- Vaak gebruikt voor “natuurlijke” materialen of onregelmatige texturen
  - Steen, hout, etc.
  - Perlin noise
- OpenGL >= 2.0



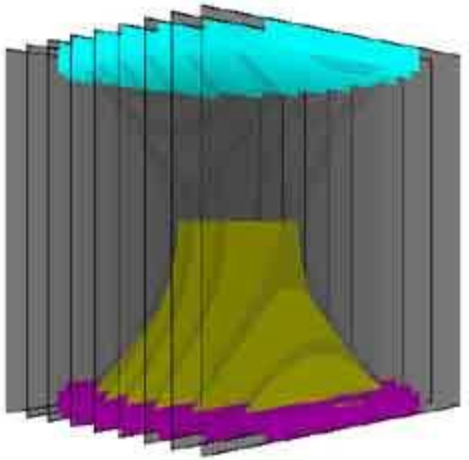
Volumetric Bump Mapping

Marble

Bump

Direct volume rendering

- 2D
  - dataset decomposed into orthographic slices along axis of volume most parallel to the viewing direction
  - move through slices back to front downloading 2D texture into memory, projected
  - may need to interpolate between slices
  - performance based on
    - software sampling rate
    - texture download rate
    - texture scan conversion rate
- 3D
  - if enough texture memory, entire volume downloaded into texture memory
  - series of planes parallel to view plane along viewing direction generated
  - polygons generated and projected back to front
  - 3D interpolation is done on the graphics card
  - if not enough texture memory, break the dataset into subvolumes (bricks) where each brick can fit in texture memory
  - process bricks back to front, being careful of the boundaries



Direct volume rendering

- Volume rendering met 2D textures (links) en 3D textures (rechts)