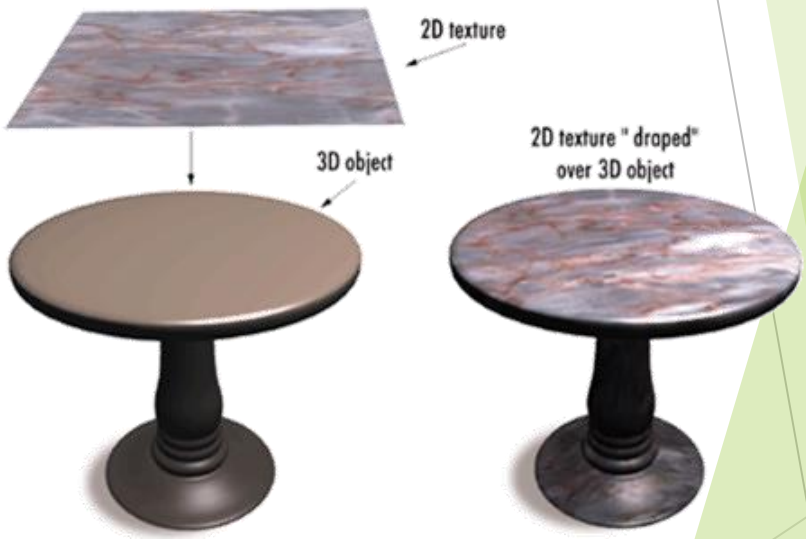
Graphics en Game Technologie

8. Texture Mapping

Robert Belleman
Computational Science Lab
Universiteit van Amsterdam
R.G.Belleman@uva.nl
(voeg a.u.b. “[GGT]” toe aan subject)

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



Source

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

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

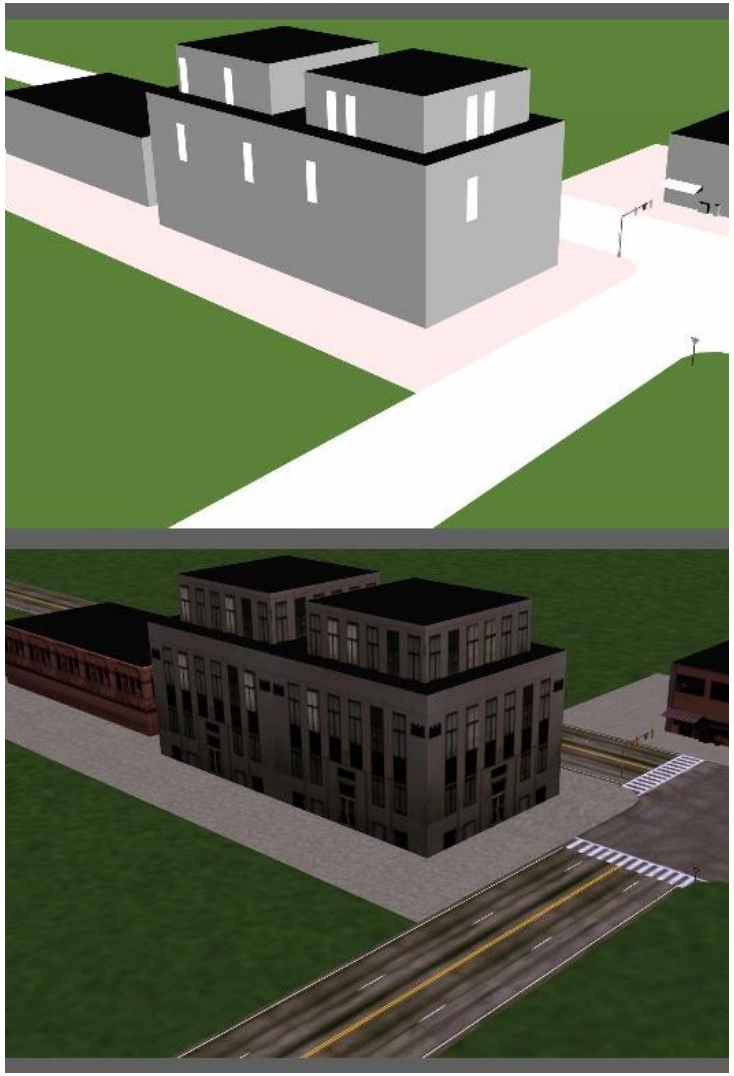
Texture mapping: waarom?

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

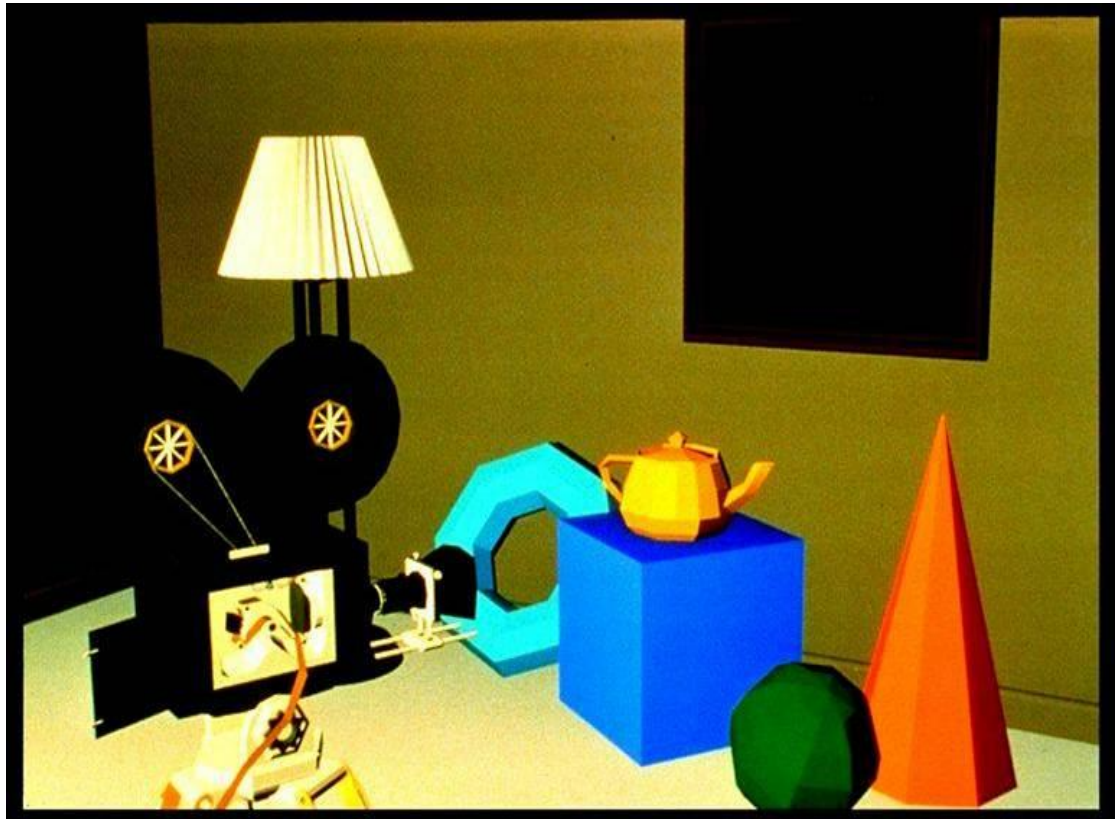
“Plaatjes plakken” op driehoeken, lijnen, punten en andere plaatjes

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, meest voorkomende)
- Volumetrische textures (3D textures)

No texture mapping

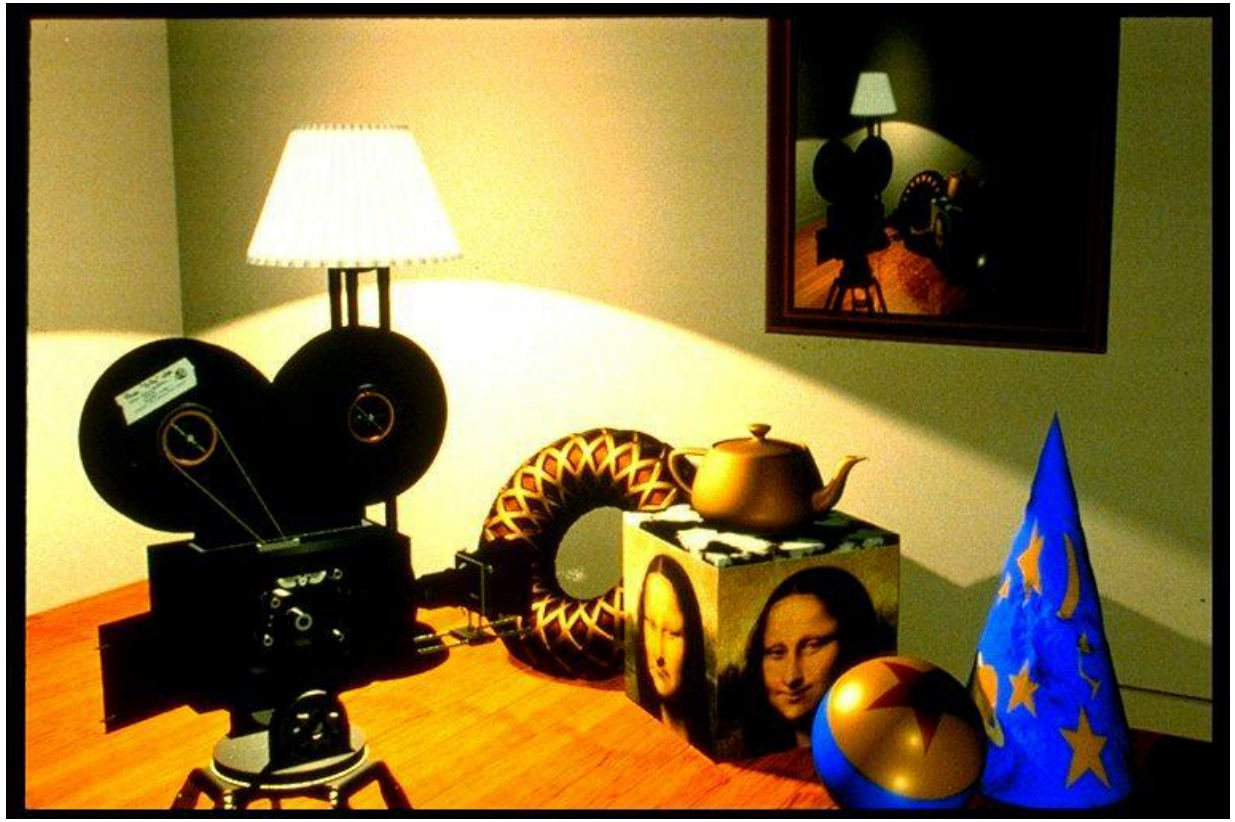


Source: Foley, van Dam et al., Computer Graphics: Principles and Practice

Texture Mapping

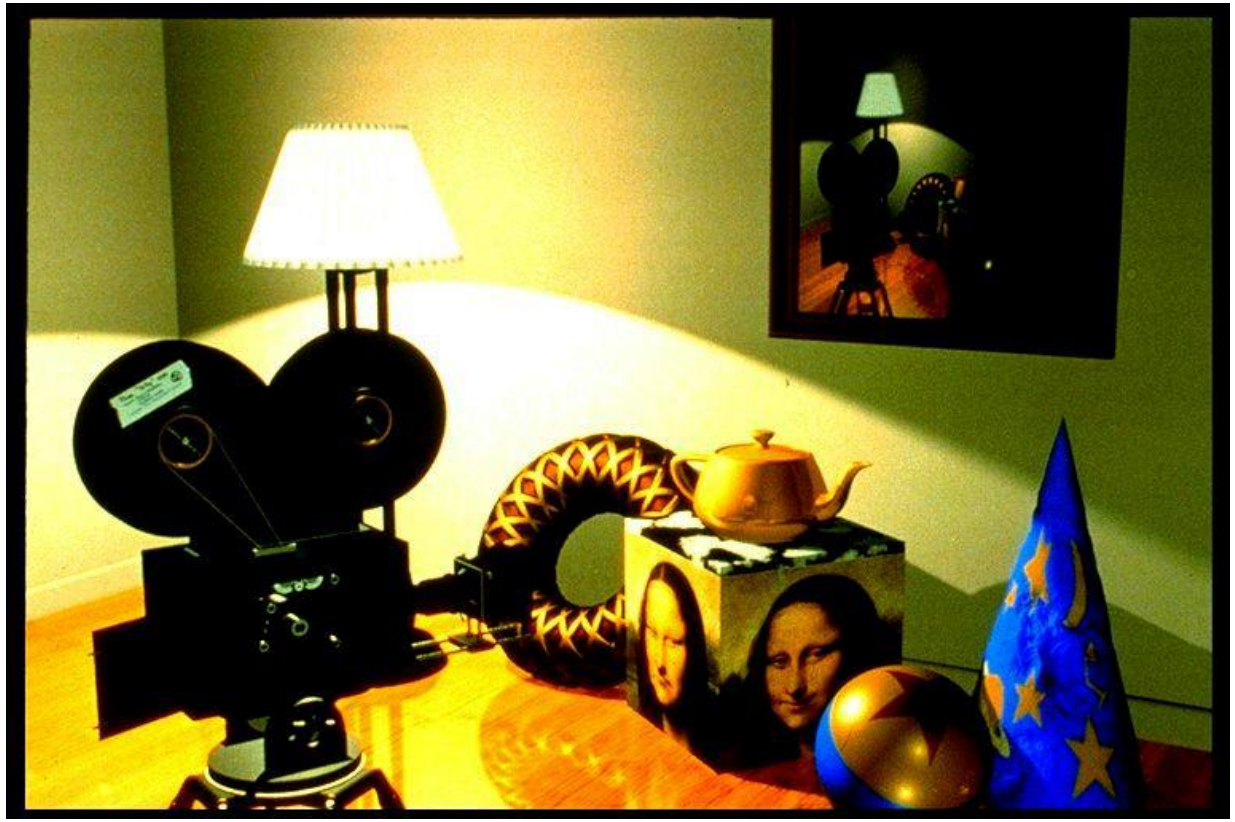
Source: Foley, van Dam et al., Computer Graphics: Principles and Practice

+ 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 object dat wordt getextured
2. Het texture image: een plaatje
3. De mapping van object coördinaten naar texture coördinaten
4. Het sampling mechanisme
5. De toepassing van de resulterende waarde(n)

Grondbeginselen

Texture image is een 2D array van (kleur)waarden (“texels”), bv:

- een plaatje
- een functie/procedure (“procedural texture”)

Texture mapping: definieer voor elk punt in een object $(x, y, z, w)_S$ een texture coördinaat $(u, v)_T$ in het plaatje

- Tussenliggende texels $(u, v)_T$ worden opgezocht door interpolatie
- De kleurwaarde in de texture wordt gebruikt om de kleur van een pixel op het scherm te bepalen

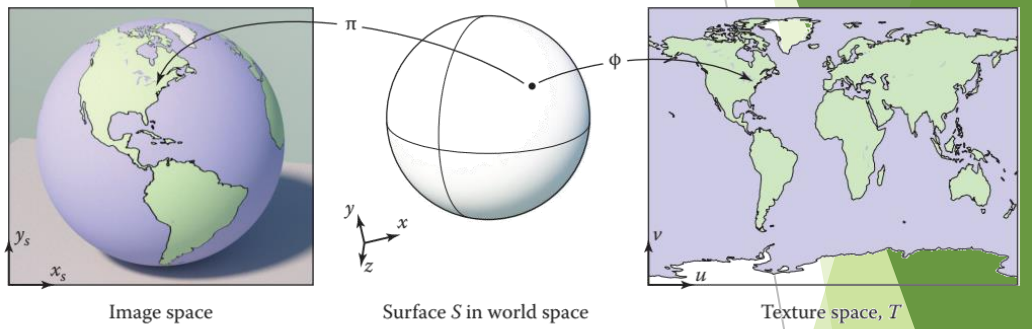
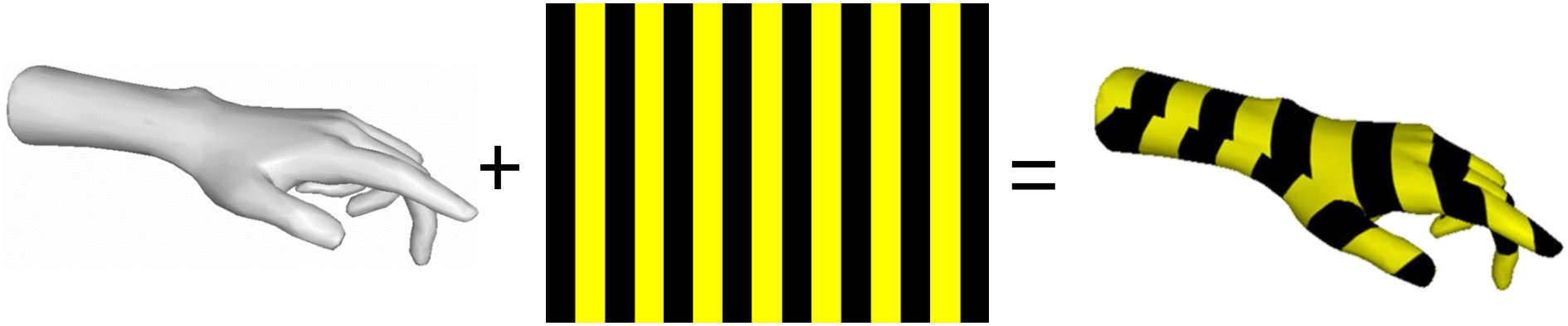


Figure 11.1. Just like the viewing projection π maps every point on an object's surface, S, to a point in the image, the texture coordinate function ϕ maps every point on the object's surface to a point in the texture map, T. Appropriately defining this function ϕ is fundamental to all applications of texture mapping.

```
Color texture_lookup(Texture t, float u, float v) {
        int i = round(u * t.width() - 0.5)
        int j = round(v * t.height() - 0.5)
        return t.get_pixel(i,j)
}
Color shade_surface_point(Surface s, Point p, Texture t) {
        Vector normal = s.get_normal(p)
        (u,v) = s.get_texcoord(p)
        Color diffuse_color = texture_lookup(u,v)
        // compute shading using diffuse_color and normal
        // return shading result
}
```

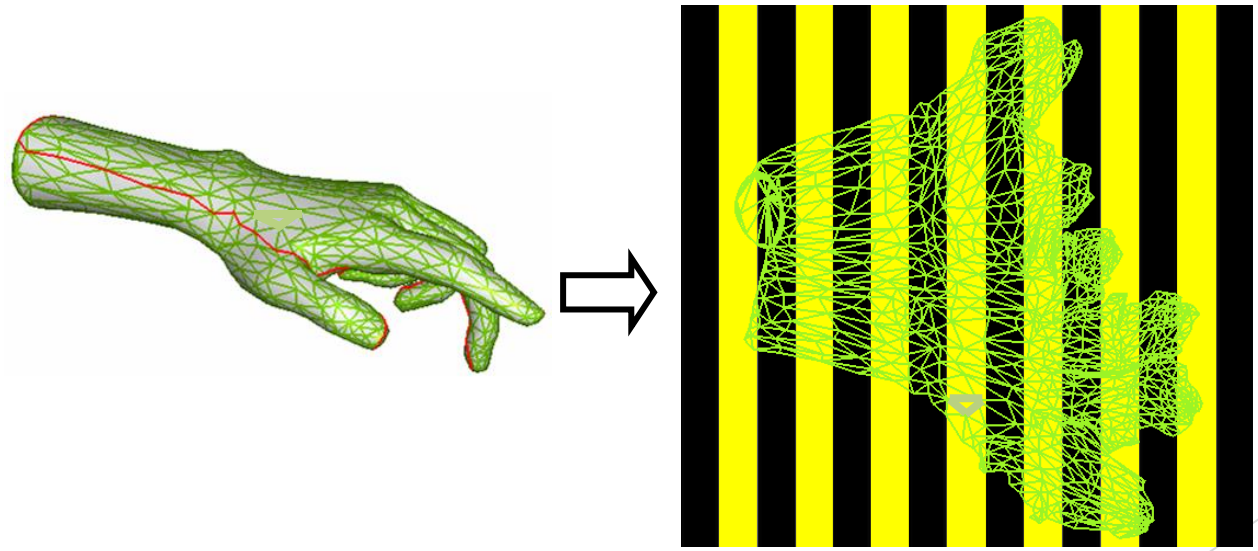
Let op: hier liggen texture coördinaten $u, v \in [0,1]$

Voorbeeld

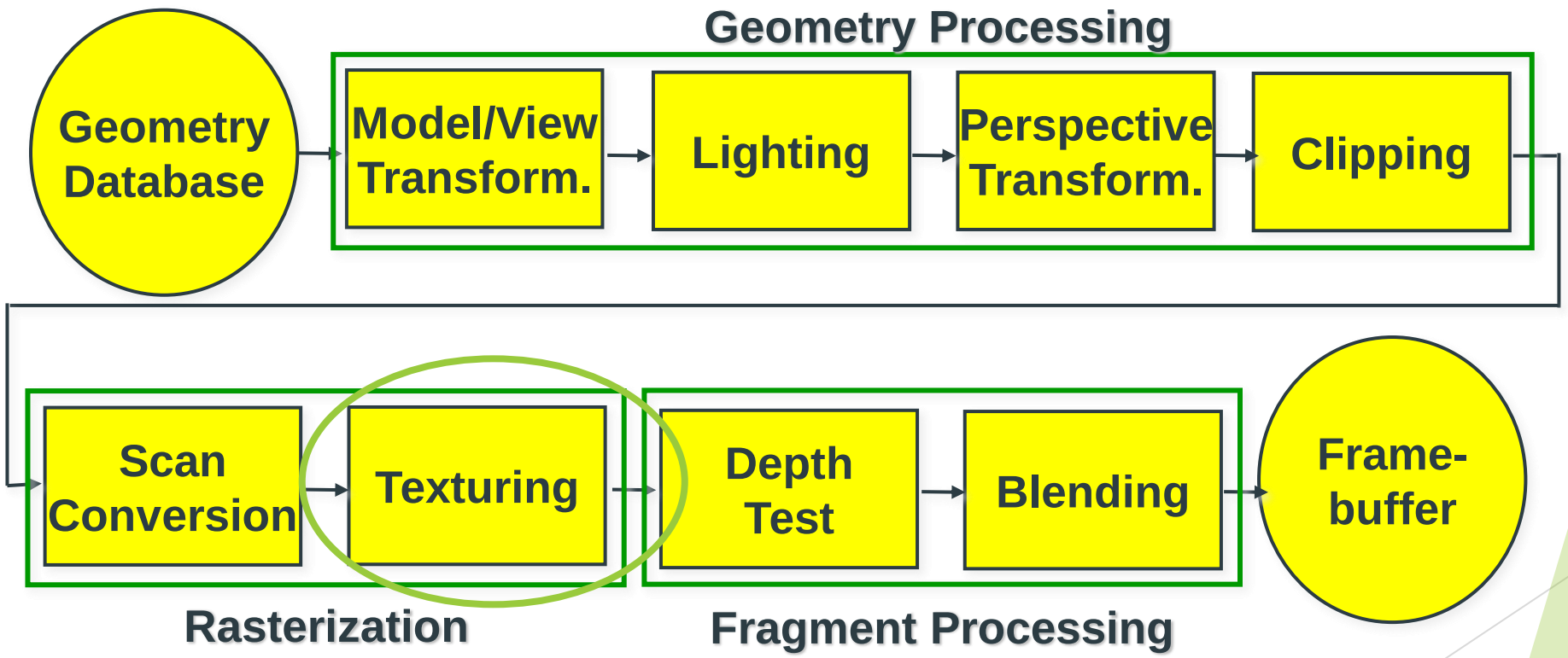


object space
"(x,y,z)" coordinates

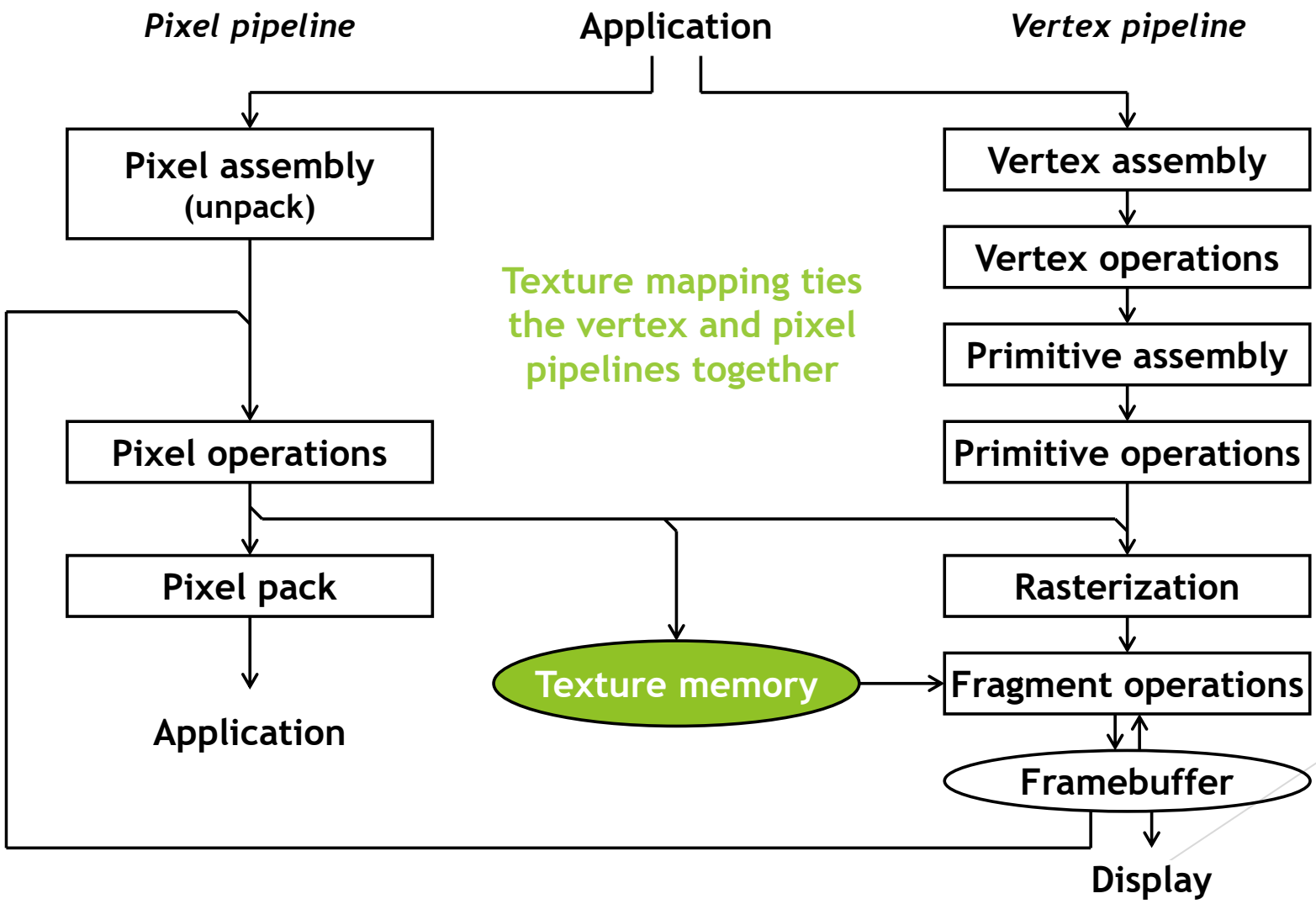
texture space
"(u,v)" coordinates
("(s,t)" coordinates when normalized)



Rendering Pipeline

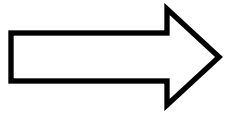


Voorbeeld: complete OpenGL pipeline

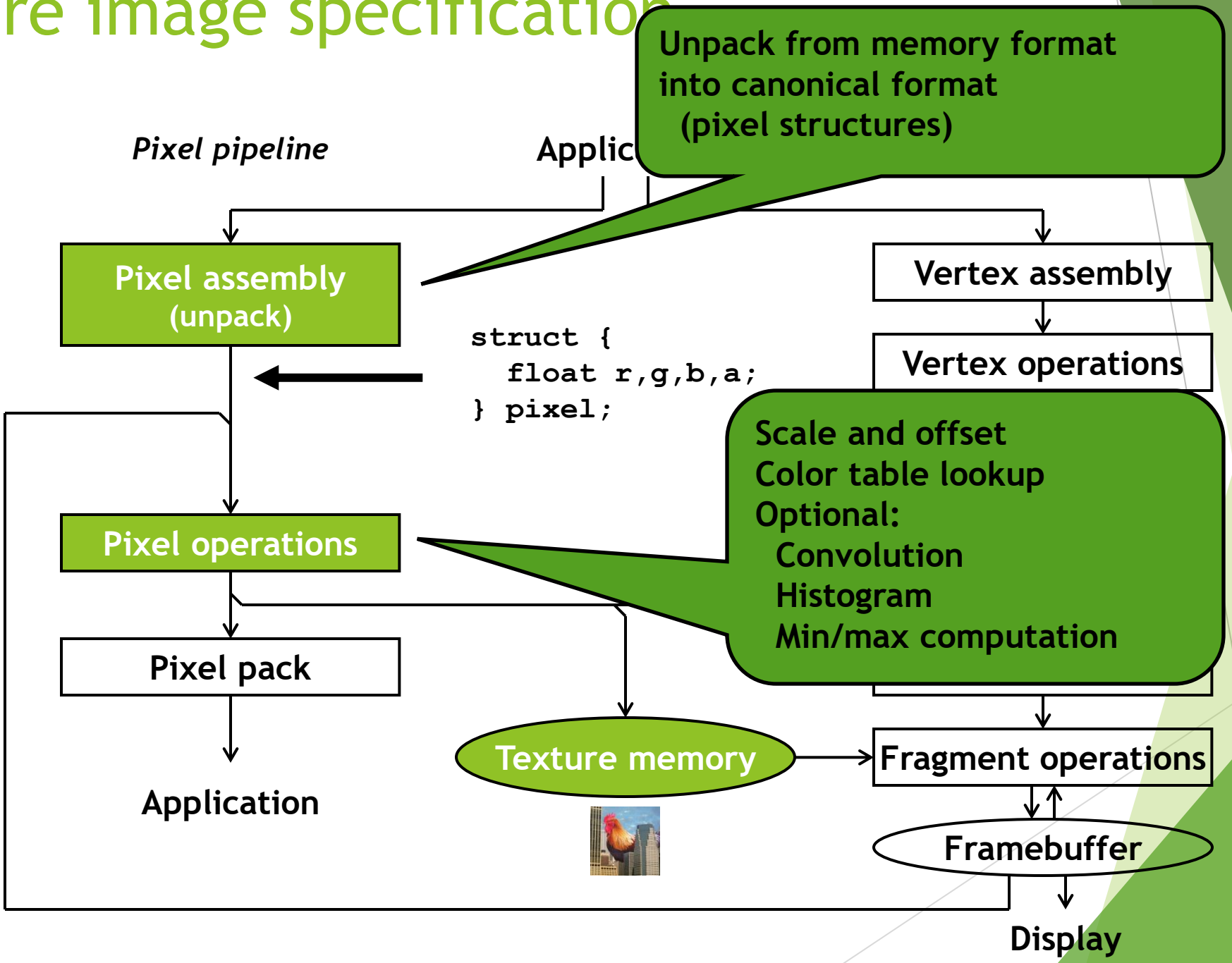


Textured quad in OpenGL

```
glGenTextures(1, &i);               // create 1 texture object
glBindTexture(GL_TEXTURE_2D, i);    // identify the texture we will use
LoadTexture("rooster", i);          // load file into texture object
glEnable(GL_TEXTURE_2D);            // enable texturing
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); // set magnification to linear interpolation
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); // set minification to linear interpolation
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);       // set texture mode to "replace"
glClearColor(1, 1, 1, 1);           // background colour: white
glClear(GL_COLOR_BUFFER_BIT);       // clear the frame buffer
glLoadIdentity();                   // clear projection matrix
glOrtho(0, 100, 0, 100, -1, 1);     // orthographic projection
glColor3f(1, 1, 1);                 // geometry colour: white
glBegin(GL_TRIANGLE_STRIP);         // two triangles, together forming a quad
  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();  // generate image into frame buffer
```

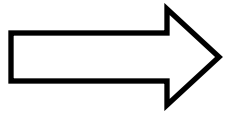


Texture image specification



Textured quad in OpenGL

```
glGenTextures(1, &i);               // create 1 texture object
glBindTexture(GL_TEXTURE_2D, i);    // identify the texture we will use
LoadTexture("rooster", i);          // load file into texture object
glEnable(GL_TEXTURE_2D);            // enable texturing
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); // set magnification to linear interpolation
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); // set minification to linear interpolation
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);       // set texture mode to "replace"
glClearColor(1, 1, 1, 1);           // background colour: white
glClear(GL_COLOR_BUFFER_BIT);       // clear the frame buffer
glLoadIdentity();                   // clear projection matrix
glOrtho(0, 100, 0, 100, -1, 1);     // orthographic projection
glColor3f(1, 1, 1);                 // geometry colour: white
glBegin(GL_TRIANGLE_STRIP);         // two triangles, together forming a quad
  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();  // generate image into frame buffer
```



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
  - N.B: Vaak is het aantal “texture units” beperkt

Texture definition in OpenGL

1. Maak een texture object vul het met image data:

```
glGenTextures(num, &identifiers)                // Maak een texture object: levert identifier
glBindTexture(GL_TEXTURE_2D, identifiers[i])    // Verbind texture commando's aan een texture object
glTexImage2D(GL_TEXTURE_2D, …)                  // Specificeer texture image (het plaatje): volgende sheet
```

2. Zet texturing aan:

```
glEnable(GL_TEXTURE_2D)
```

3. Specificeer texturing parameters:

```
glTexParameteri(GL_TEXTURE_2D, …, …)            // Specificeer hoe texture gesampled moet worden
glTexEnvf(…)                                    // Specificeer texture "mode" parameters
```

4. Specificeer texture coordinaten voor elke punt op oppervlak:

```
glTexCoord2f(0,0); glVertex3f(x,y,z);
```

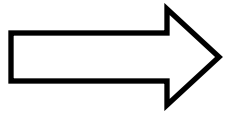
Details: glTexImage2D

Specificeren van een texture:

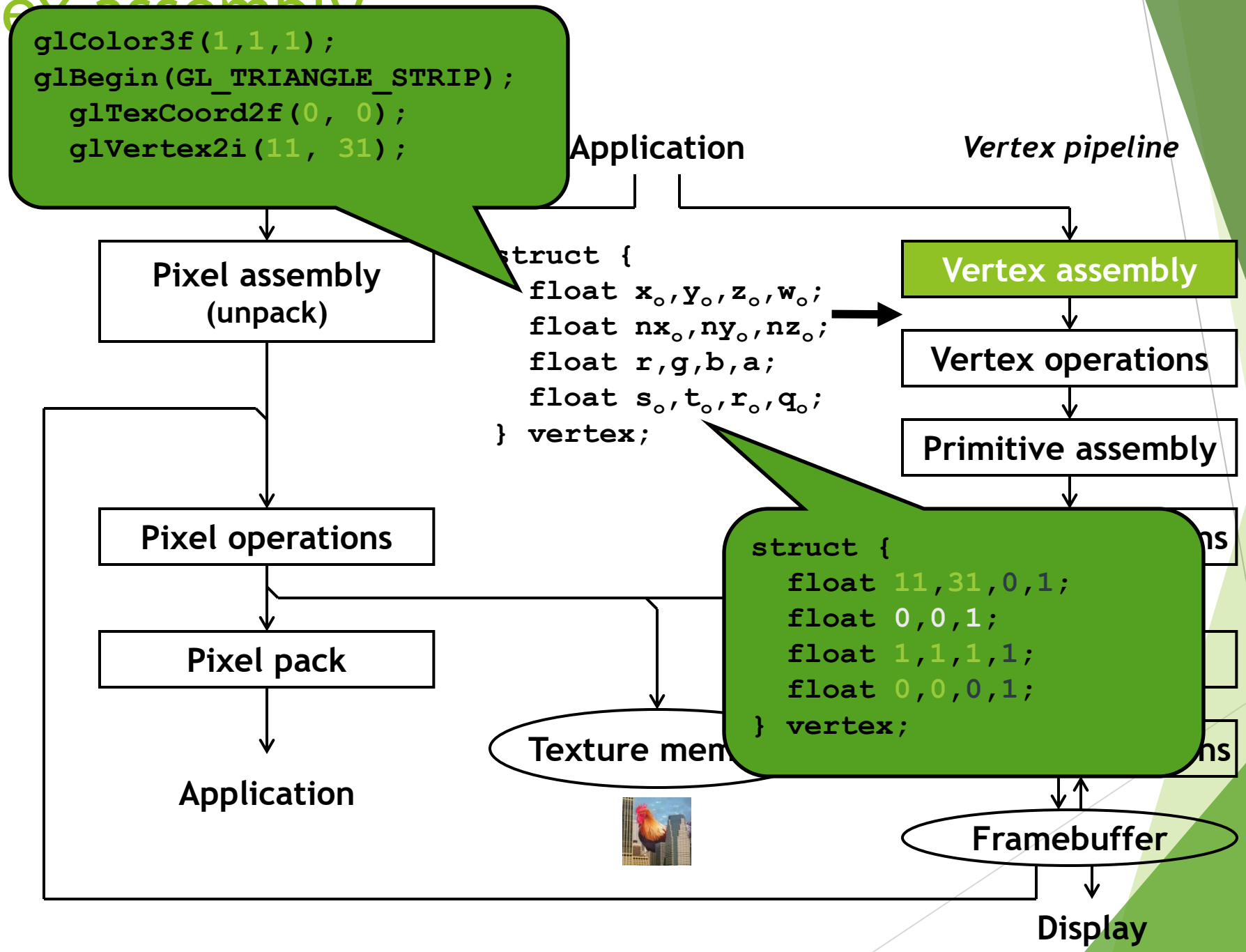
```
void glTexImage2D( GLenum target,          // type of texture to define
                   GLint level,            // mipmap level for this texture
                   GLint internalFormat,   // type of colour format to use for this texture
                   GLsizei width,          // number of horizontal pixels in the image
                   GLsizei height,         // number of vertical pixels in the image
                   GLint border,           // must be 0
                   GLenum format,          // colour format of pixel data in the image
                   GLenum type,            // data type of pixel data in the image
                   const GLvoid *pixels )  // pointer to pixel data of the image
```

Textured quad in OpenGL

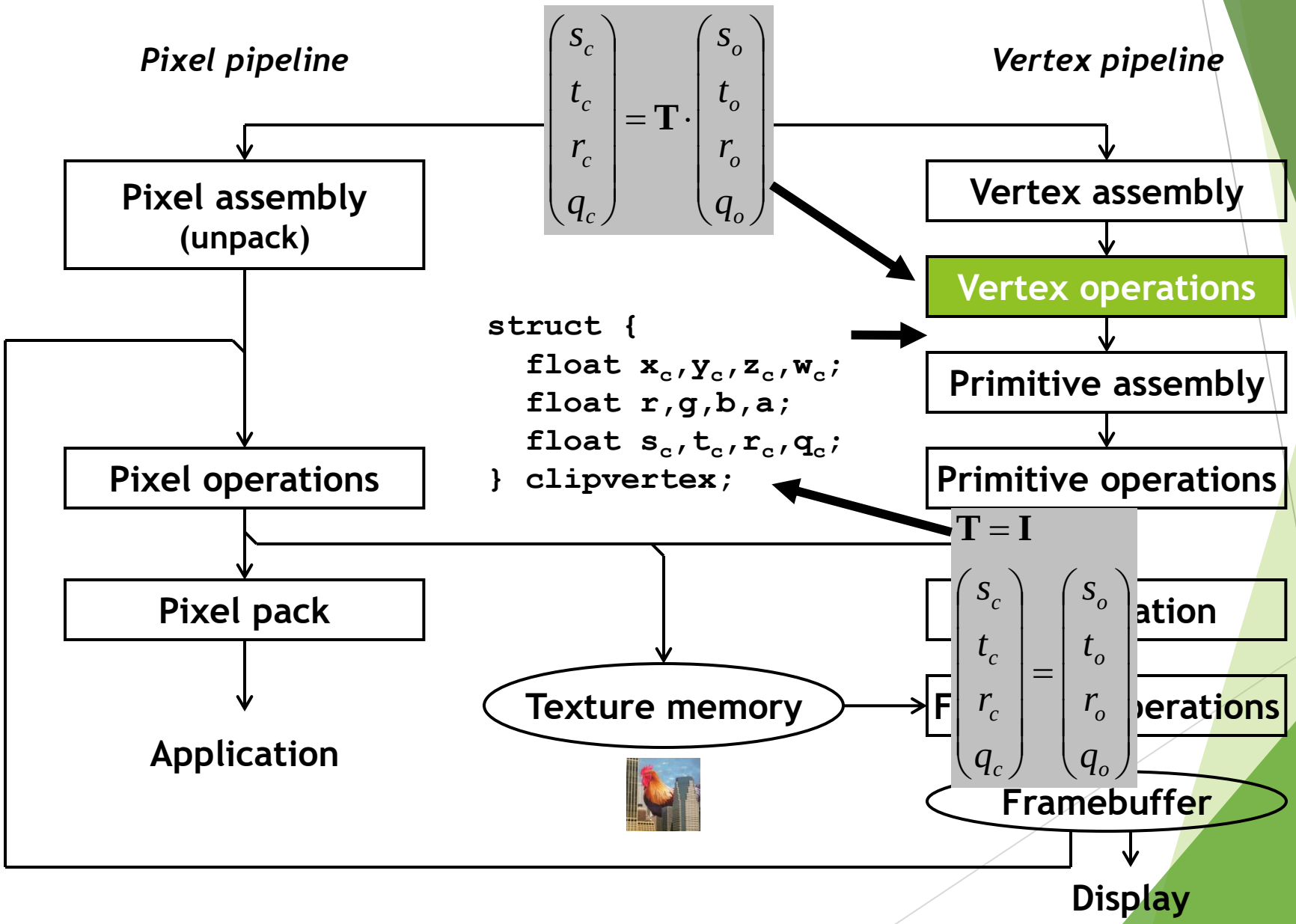
```
glGenTextures(1, &i);                // create 1 texture object
glBindTexture(GL_TEXTURE_2D, i);     // identify the texture we will use
LoadTexture("rooster", i);           // load file into texture object
glEnable(GL_TEXTURE_2D);             // enable texturing
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); // set magnification to linear interpolation
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); // set minification to linear interpolation
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);       // set texture mode to "replace"
glClearColor(1, 1, 1, 1);            // background colour: white
glClear(GL_COLOR_BUFFER_BIT);        // clear the frame buffer
glLoadIdentity();                    // clear projection matrix
glOrtho(0, 100, 0, 100, -1, 1);      // orthographic projection
glColor3f(1, 1, 1);                  // geometry colour: white
glBegin(GL_TRIANGLE_STRIP);          // two triangles, together forming a quad
  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();  // generate image into frame buffer
```



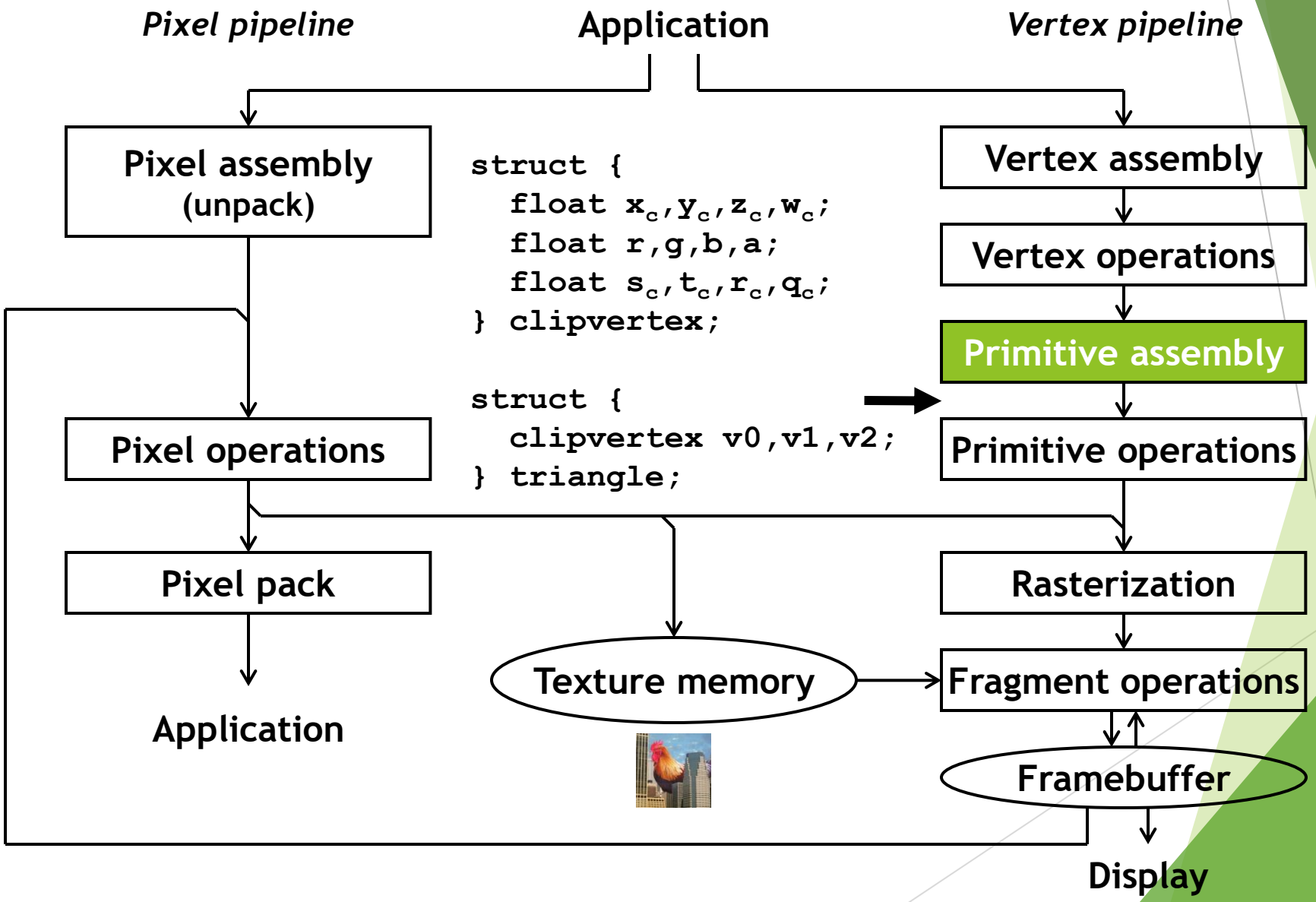
Vertex assembly



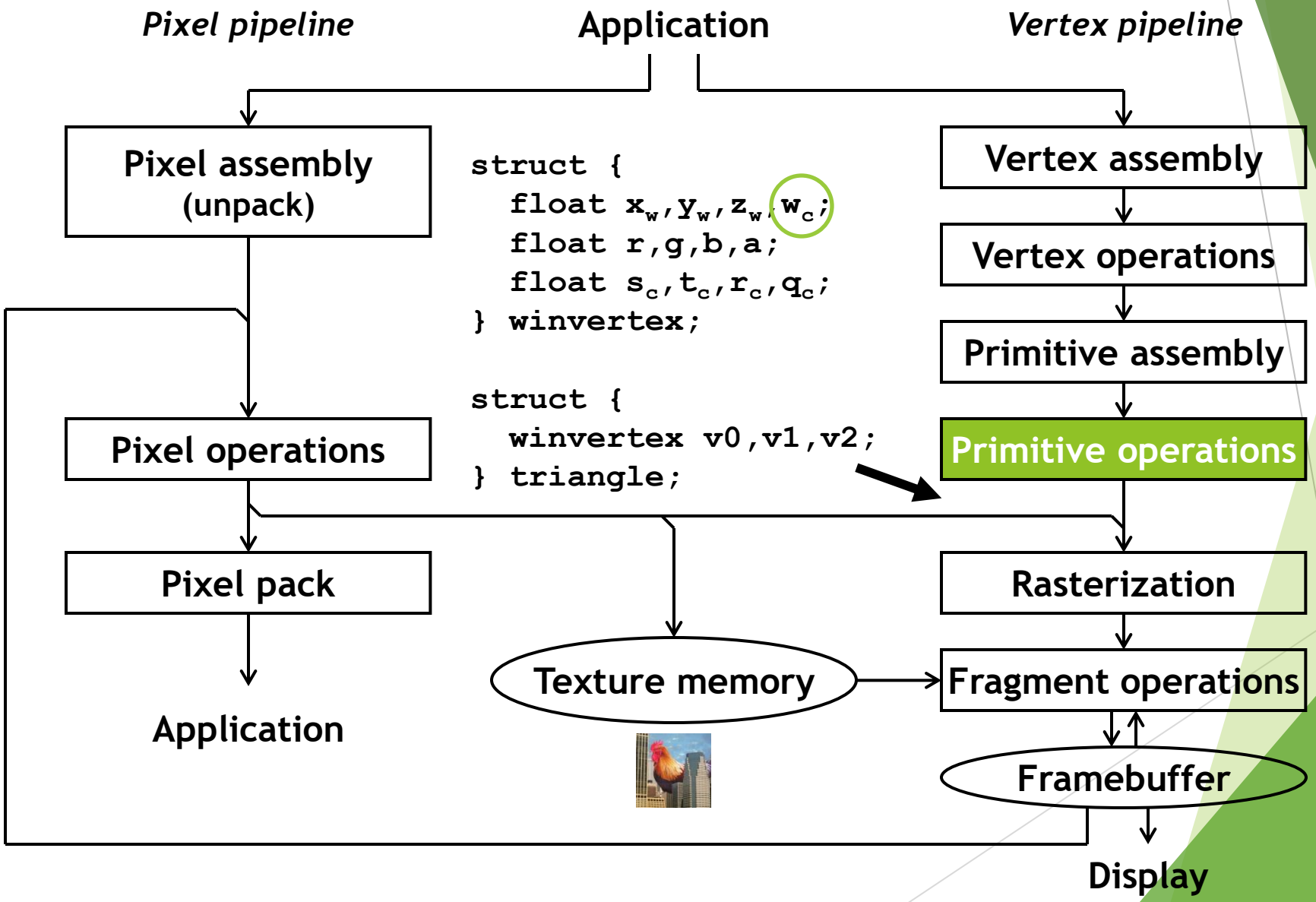
Vertex operations



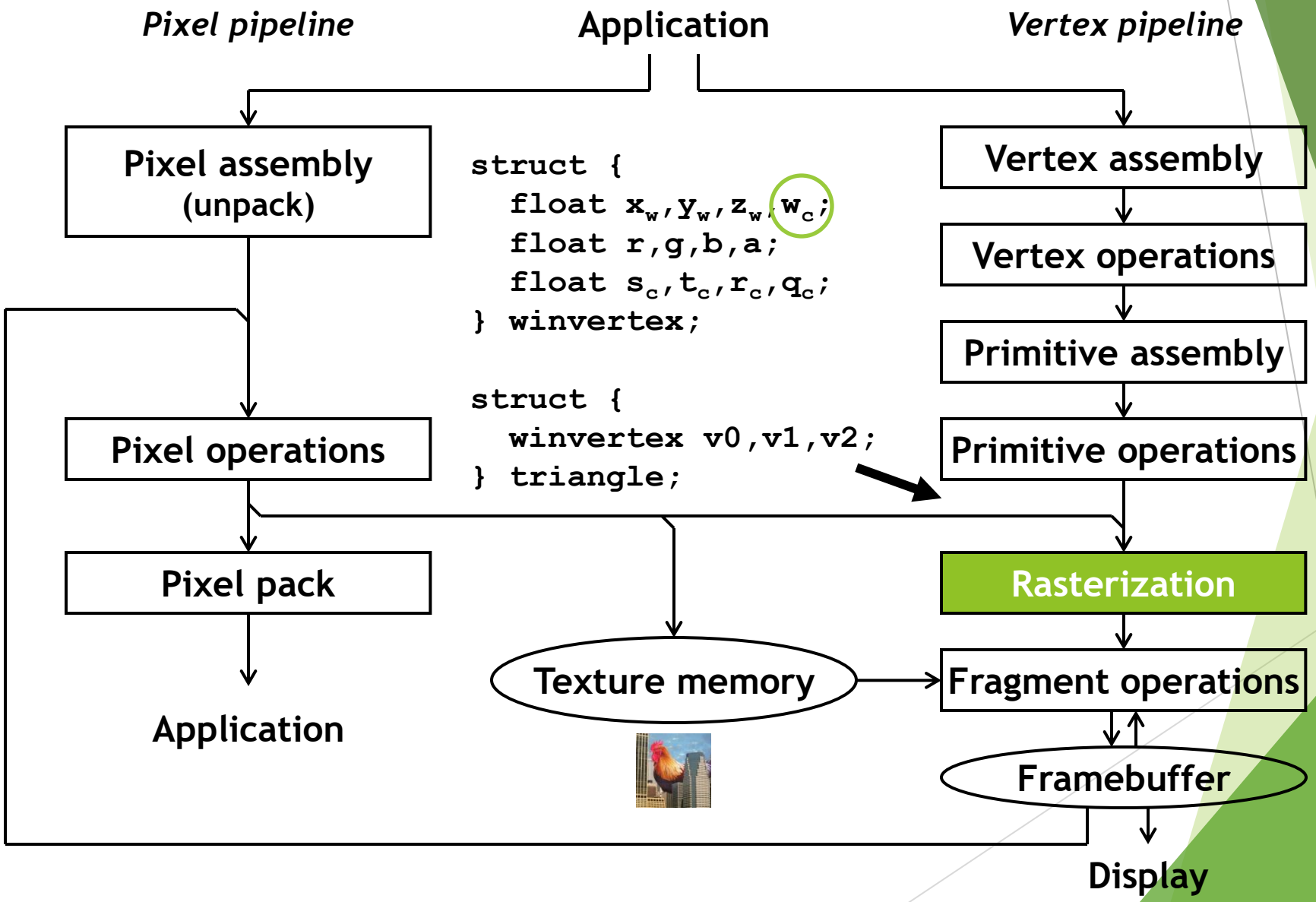
Primitive assembly



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”)coördinaten gaat dat niet!

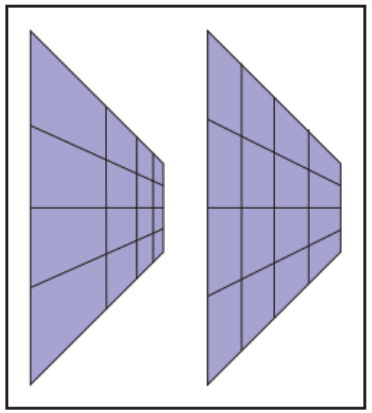
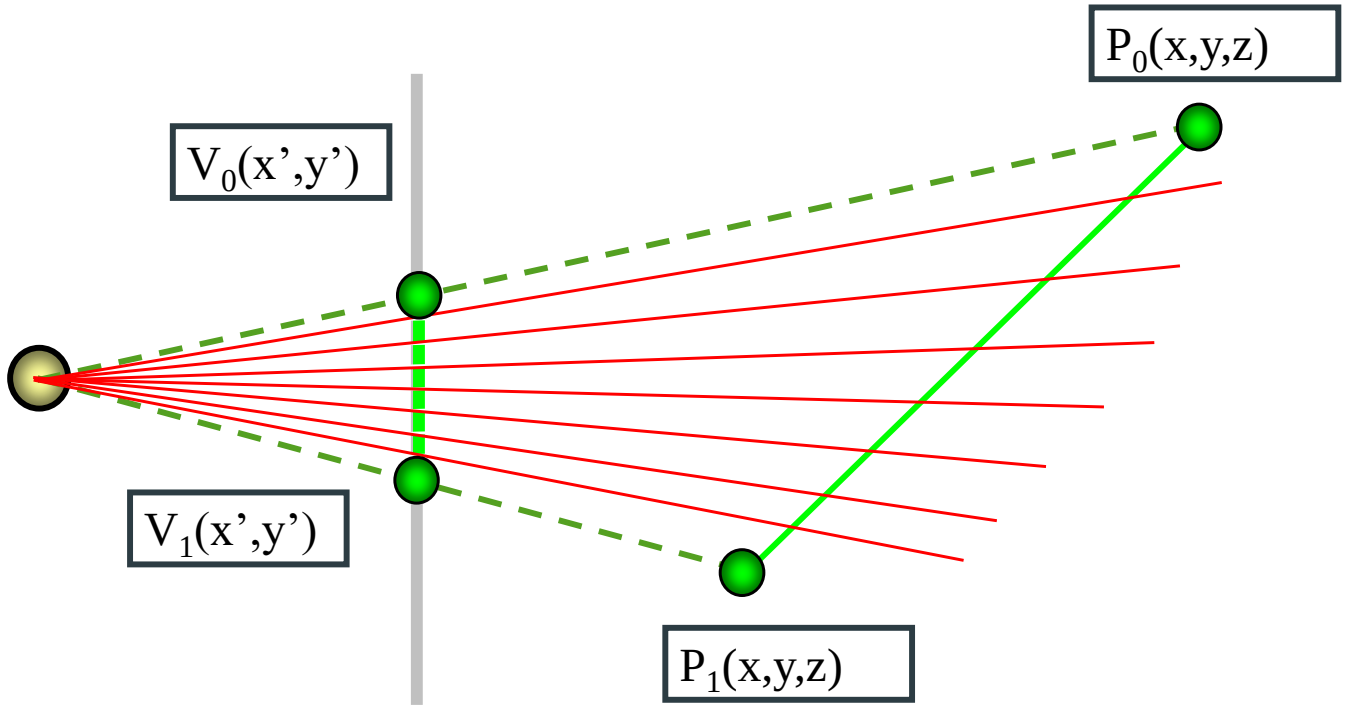


Figure 11.15. Left: correct perspective. Right: interpolation in screen space.

Interpolatie: scherm- vs. wereldcoördinaten

Schermcoördinaten interpolatie incorrect

- Geen probleem wordt bij Gouraud shading, maar artefacten worden opeens goed zichtbaar bij texturing



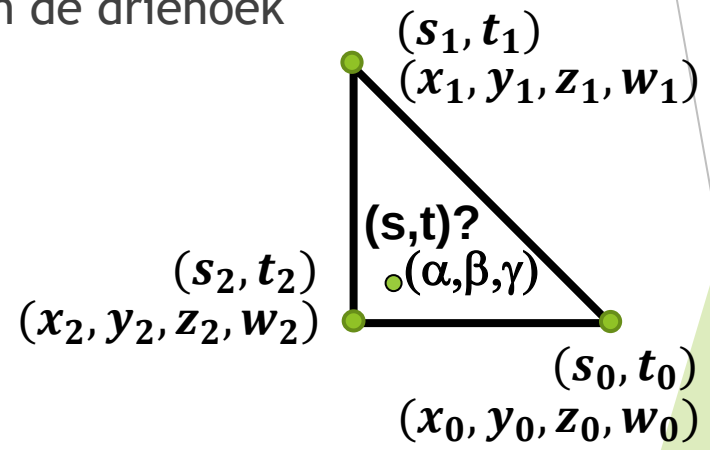
Perspective division

Interpolatie met correct perspectief: “perspective division”:

- Bereken barycentrische coördinaten α, β, γ van punt in de driehoek
- Bereken texture coördinaten (s, t) met:

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

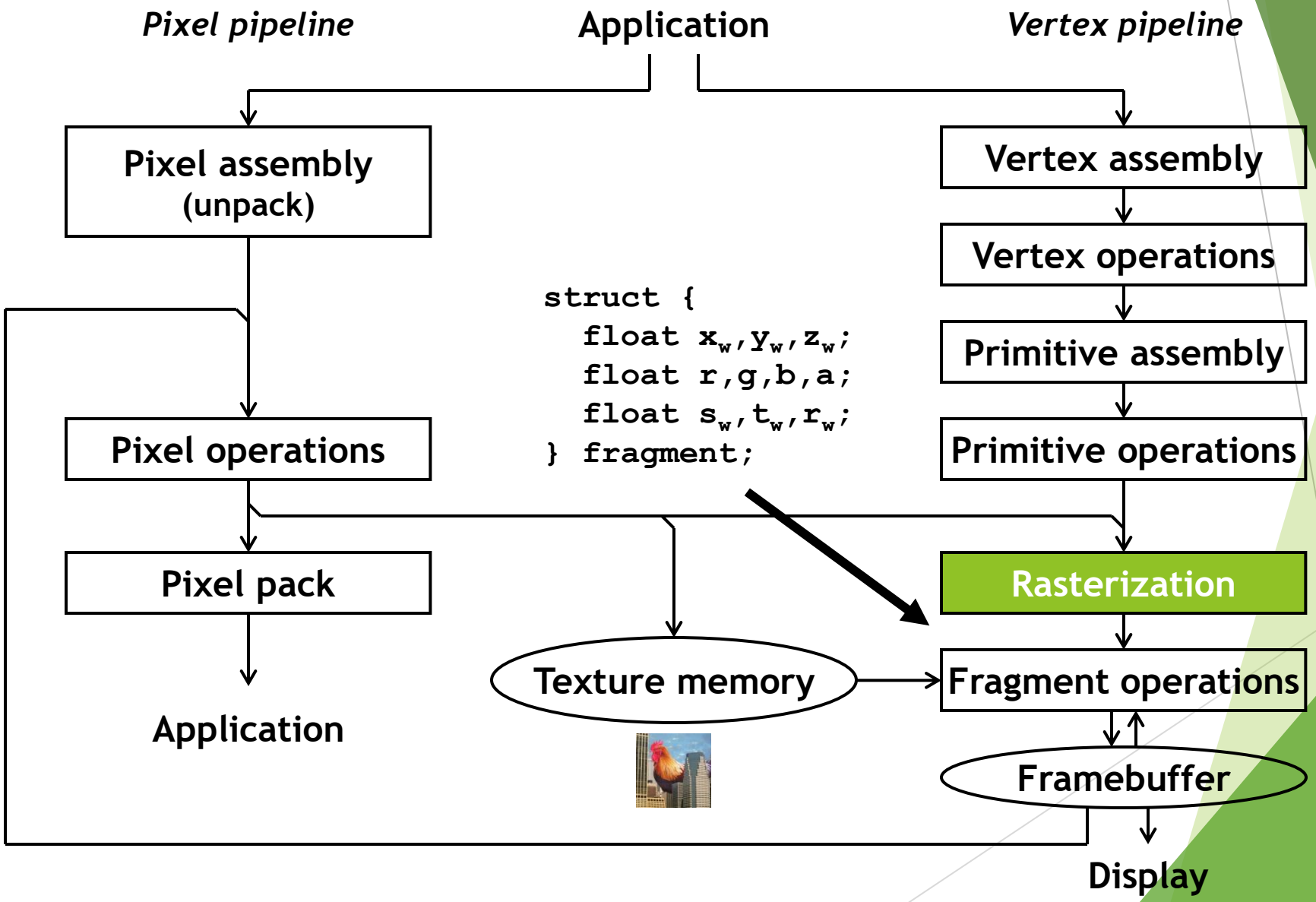
$$t = \frac{\alpha \cdot t_0/w_0 + \beta \cdot t_1/w_1 + \gamma \cdot t_2/w_2}{\alpha/w_0 + \beta/w_1 + \gamma/w_2}$$



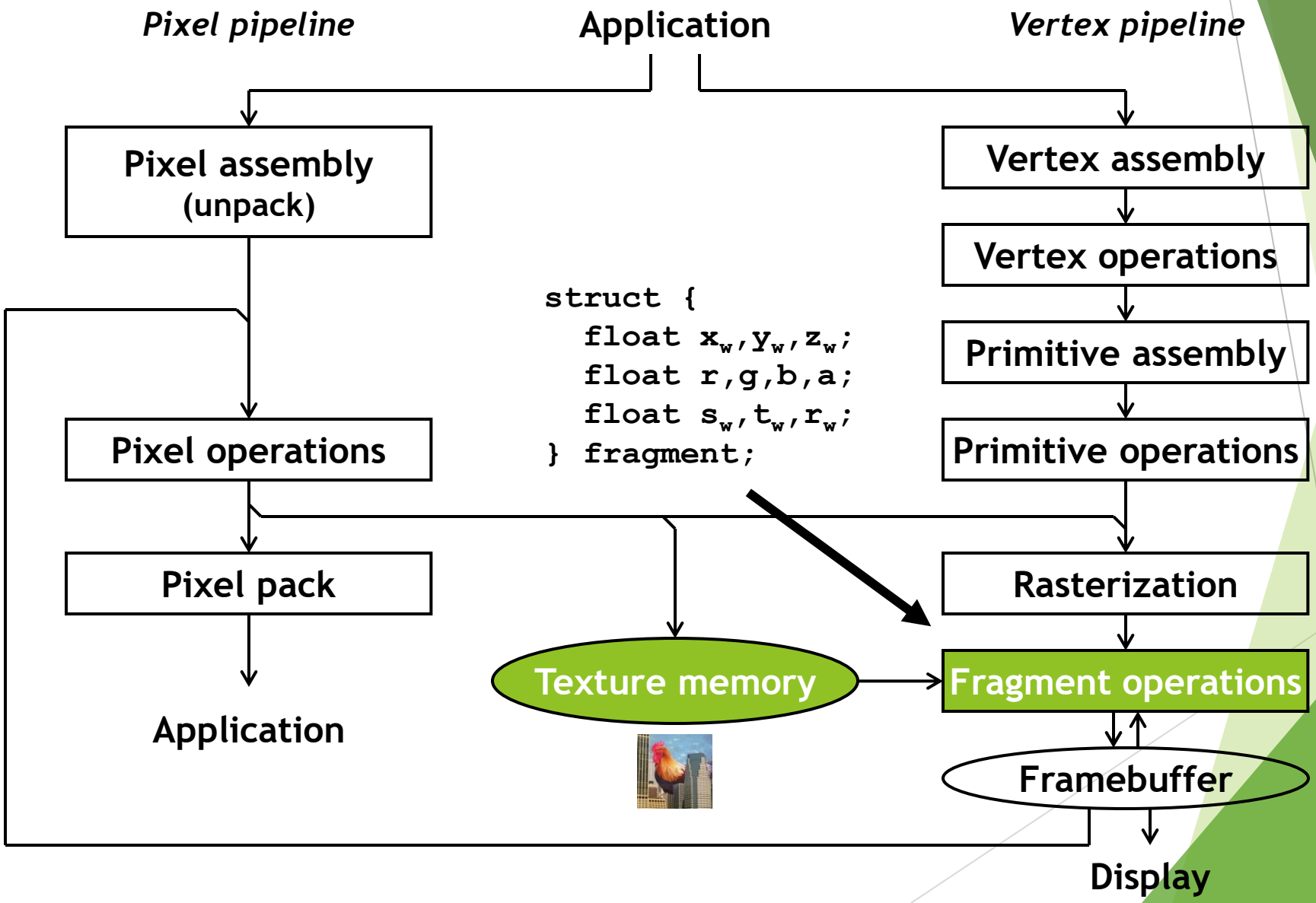
waar:

- $(s_0, t_0), (s_1, t_1), (s_2, t_2)$: texture coördinaten van de hoekpunten
- w_0, w_1, w_2 : homogene coördinaten van de hoekpunten

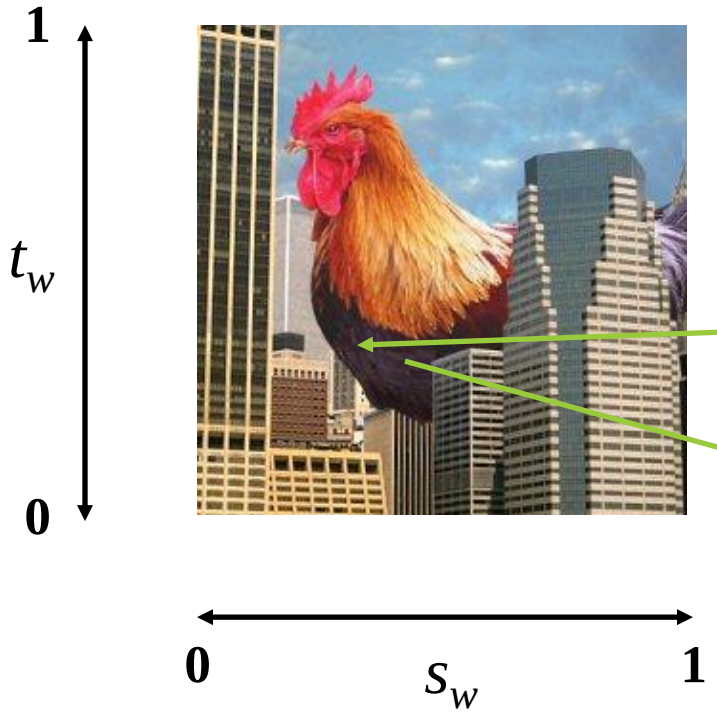
Rasterization



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

Let op: hier liggen texture coördinaten $s, t \in [0,1]$

Texture application

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 xw,yw,zw;
  float r',g',b',a';
} fragment;
```

Texture “modes”

Hoe wordt de waarde uit het texture toegepast op het oppervlak?

- 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

Texturing in OpenGL

1. Maak een texture object vul het met image data:

```
glGenTextures(num, &identifiers)                // Maak een texture object: levert identifier
glBindTexture(GL_TEXTURE_2D, identifiers[i])    // Verbind texture commando's aan een texture object
glTexImage2D(GL_TEXTURE_2D, …)                  // Specificeer texture image (het plaatje): volgende sheet
```

2. Zet texturing aan:

```
glEnable(GL_TEXTURE_2D)
```

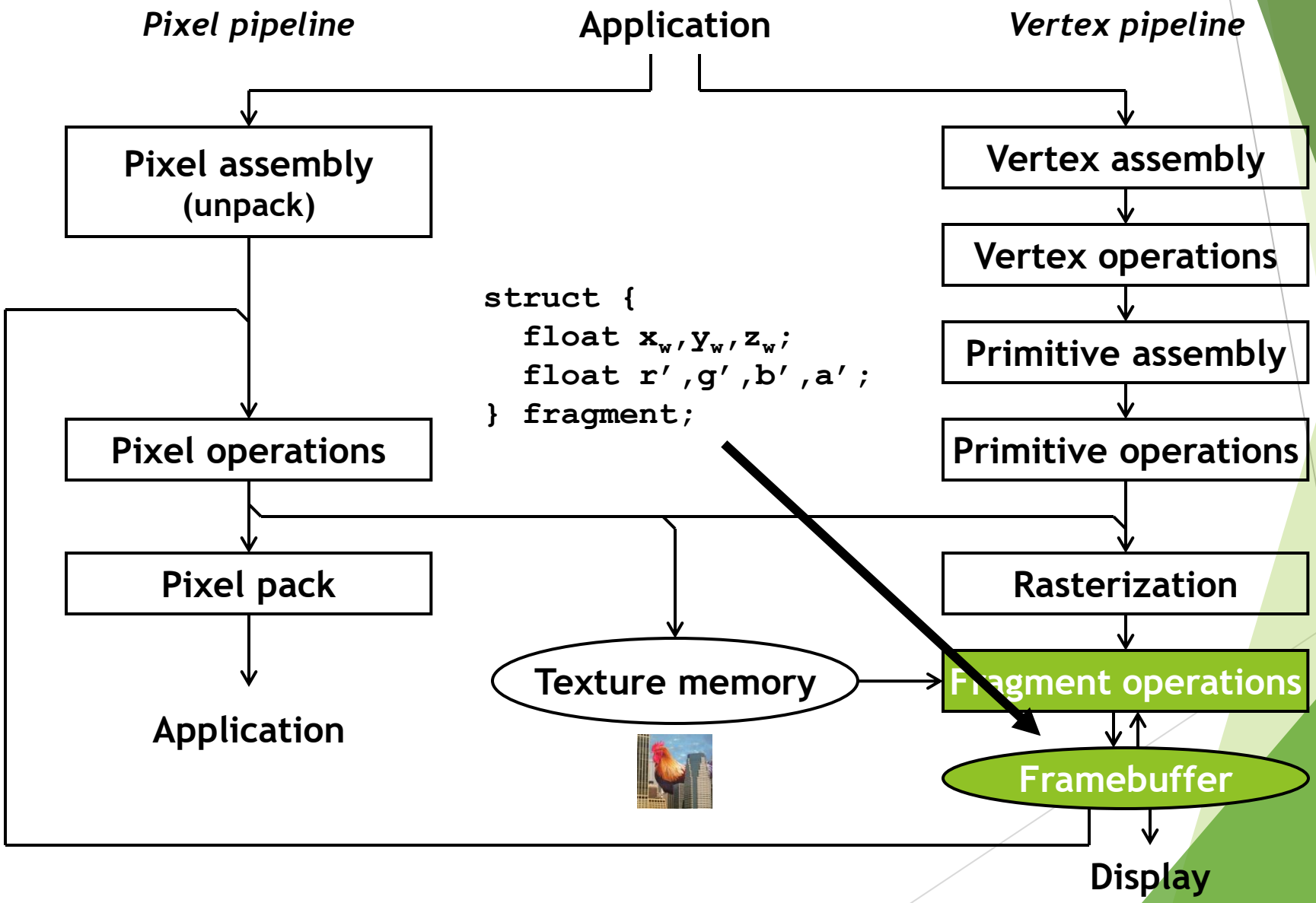
3. Specificeer texturing parameters:

```
glTexParameteri(GL_TEXTURE_2D, …, …)            // Specificeer hoe texture gesampled moet worden
glTexEnvf(…)                                    // Specificeer texture "mode" parameters
```

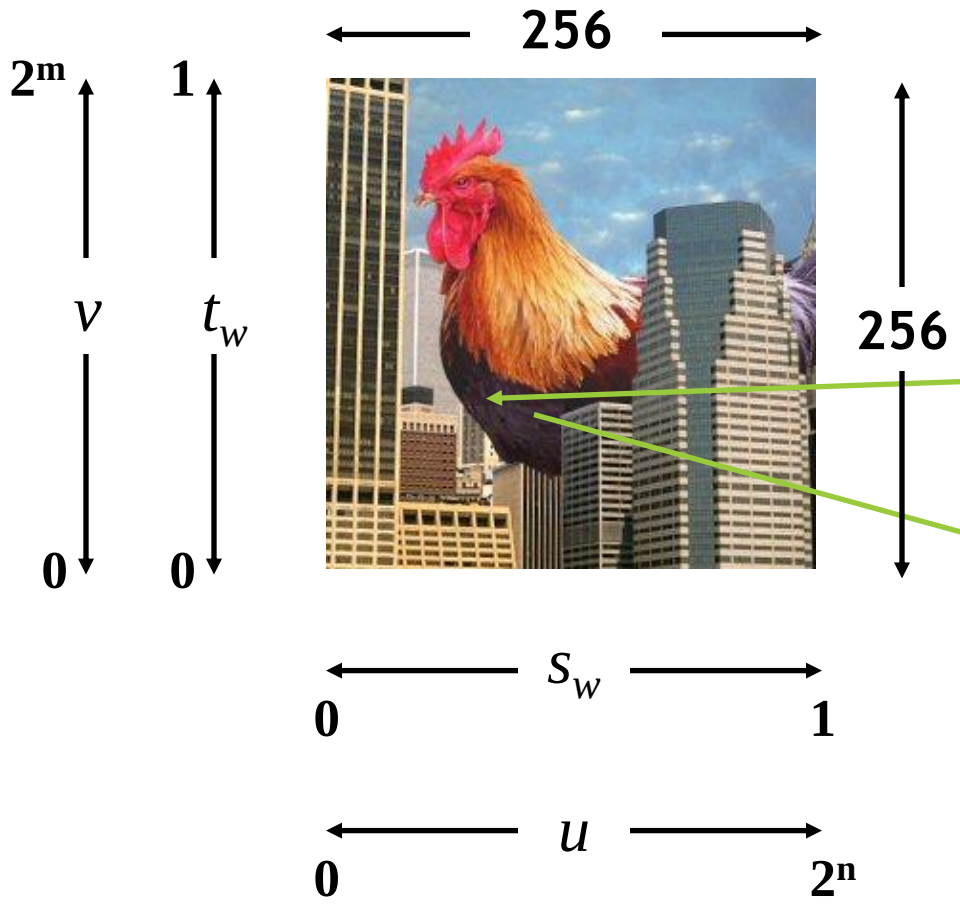
4. Specificeer texture coordinaten voor elke punt op oppervlak:

```
glTexCoord2f(0,0); glVertex3f(x,y,z);
```

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;
```

Let op: hier liggen texture coördinaten $s, t \in [0,1]$
En texel coördinaten $u \in [0,2^n], v \in [0,2^m]$

For this image n = m = 8

Texture Lookup: tiling and clamping

Wat gebeurt er als s of t buiten het interval $[0,1]$ vallen?

De volgende opties:

1. 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);
```

2. 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);
```

Fractional Texture Coordinates

texture image

(0,1) (1,1)

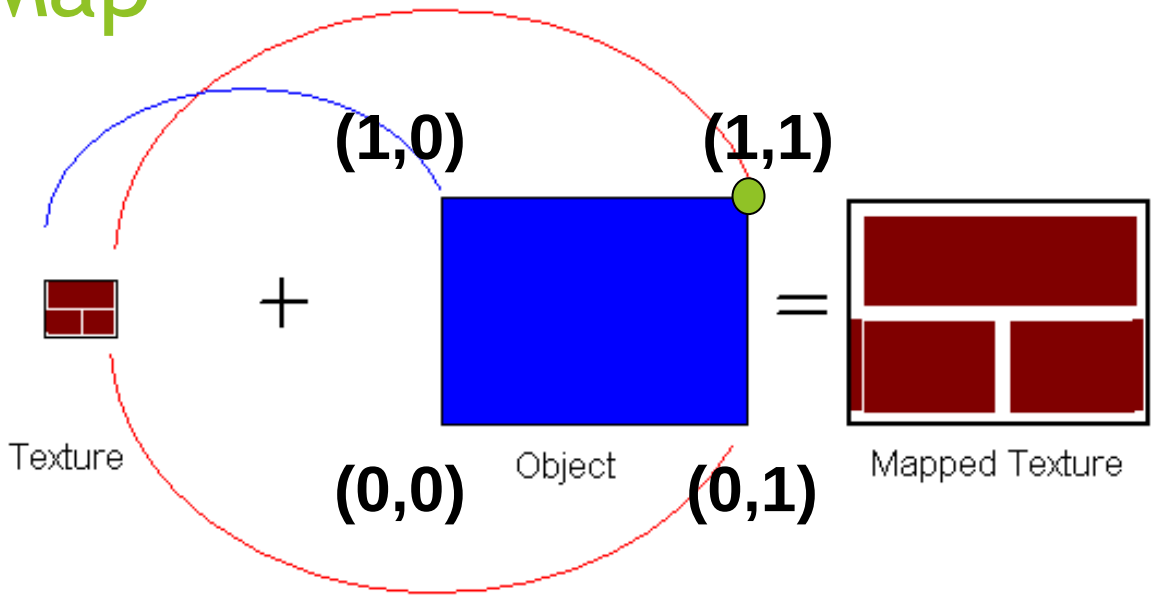
(0,0) (1,0)

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

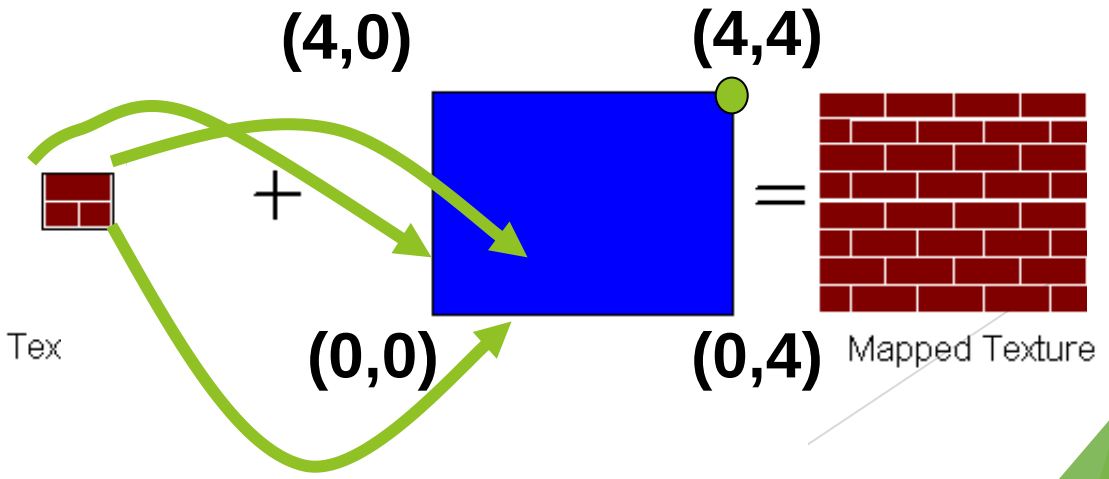
(0,0) (.25,0)

Tiled Texture Map

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

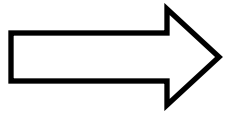



```
glTexCoord2d(4, 4);
glVertex3d (x, y, z);
```



Textured quad in OpenGL

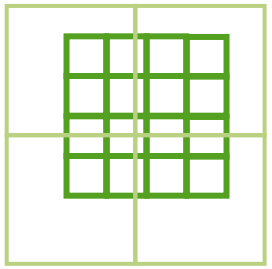
```
glGenTextures(1, &i);               // create 1 texture object
glBindTexture(GL_TEXTURE_2D, i);    // identify the texture we will use
LoadTexture("rooster", i);          // load file into texture object
glEnable(GL_TEXTURE_2D);            // enable texturing
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); // set magnification to linear interpolation
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); // set minification to linear interpolation
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);       // set texture mode to "replace"
glClearColor(1, 1, 1, 1);           // background colour: white
glClear(GL_COLOR_BUFFER_BIT);       // clear the frame buffer
glLoadIdentity();                   // clear projection matrix
glOrtho(0, 100, 0, 100, -1, 1);     // orthographic projection
glColor3f(1, 1, 1);                 // geometry colour: white
glBegin(GL_TRIANGLE_STRIP);         // two triangles, together forming a quad
  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();  // generate image into frame buffer
```



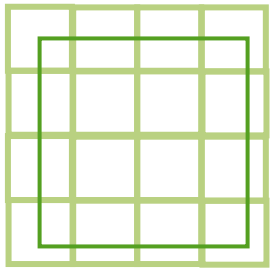
Texture filtering

Hoe ga je om met:

- pixels die veel kleiner zijn dan een texel?
  - Weinig texel informatie per pixel
  - Magnification filtering: “upsampling”: nearest neighbour, linear interpolation



- pixels die veel groter zijn dan een texel?
  - Veel texel informatie per pixel
  - Minification filtering: “downsampling”: nearest neighbour, linear interpolation, nearest mipmap nearest neighbour nearest mipmap linear, linear mipmap nearest neighbour, linear mipmap linear



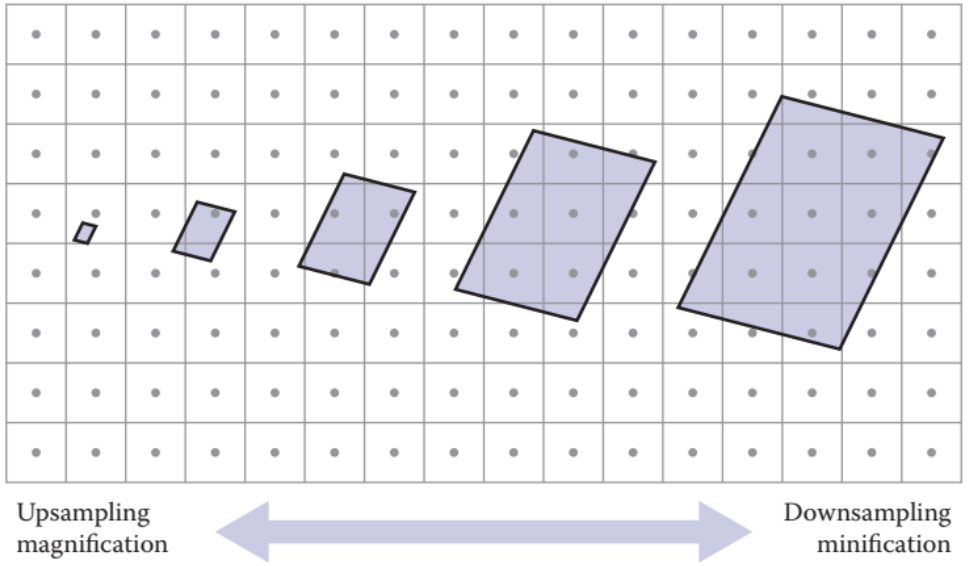
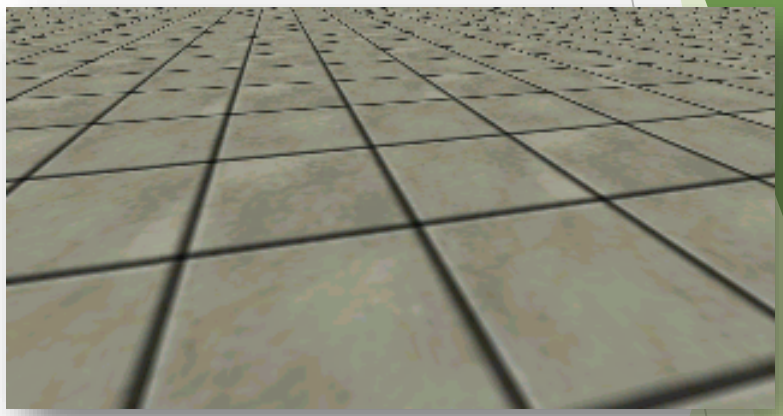


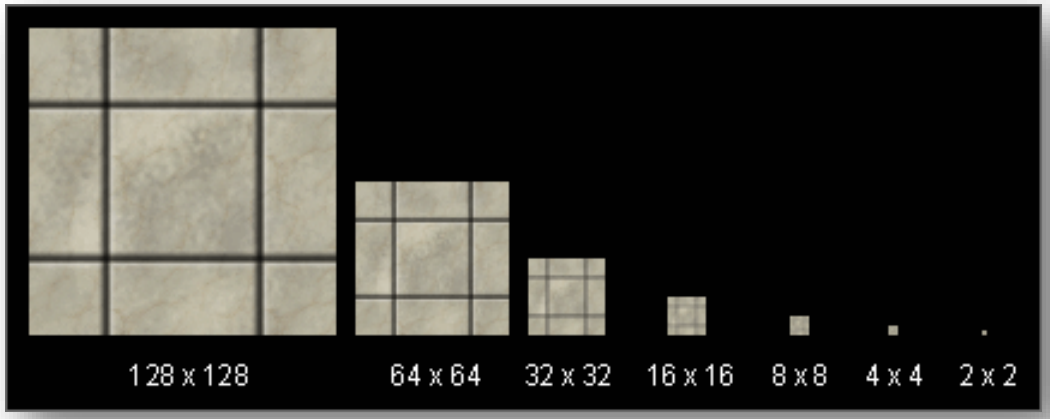
Figure 11.20. The dominant issues in texture filtering change with the footprint size. For small footprints (left) interpolating between pixels is needed to avoid blocky artifacts; for large footprints, the challenge is to efficiently find the average of many pixels.

MIPmapping

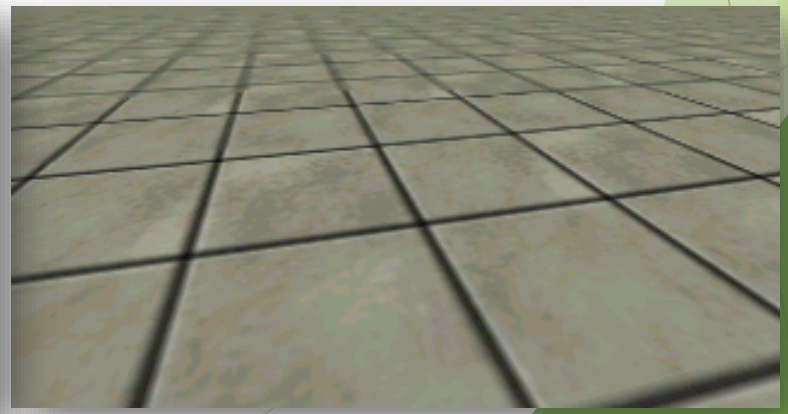
- Gebruikt een “image pyramid” met op voorhand berekende verkleinde en gefilterde versies van de texture
- De hele “image pyramid” zit in een blok geheugen



Texturing zonder MIP-mapping



MIPmap image pyramid



Texturing met MIP-mapping

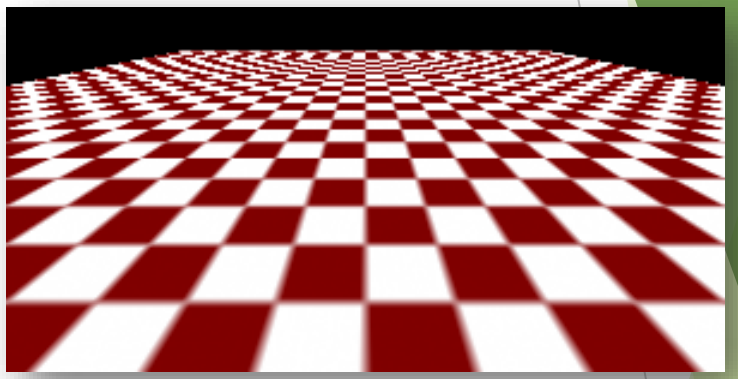
MIPmaps

MIP: “multum in parvo” -- many things in a small place

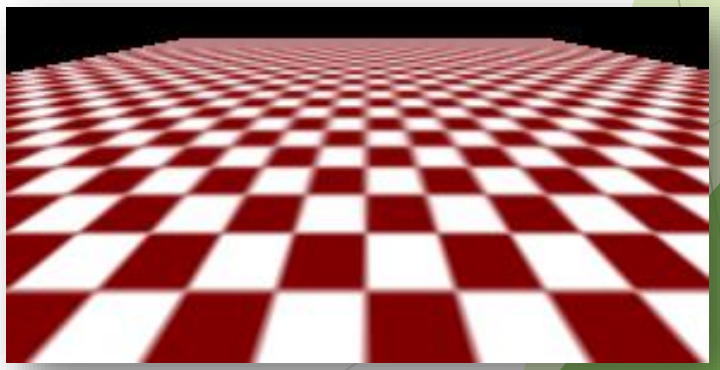
- Specificeer een serie van texture maps met afnemende resolutie
- Meer geheugenruimte nodig (slechts 1/3 meer; zie volgende sheet)
- Voorkomt flikkeren als objecten bewegen

`gluBuild2DMipmaps()` functie

- Maakt automatisch een serie textures van de oorspronkelijke texture grootte tot 1x1 pixel



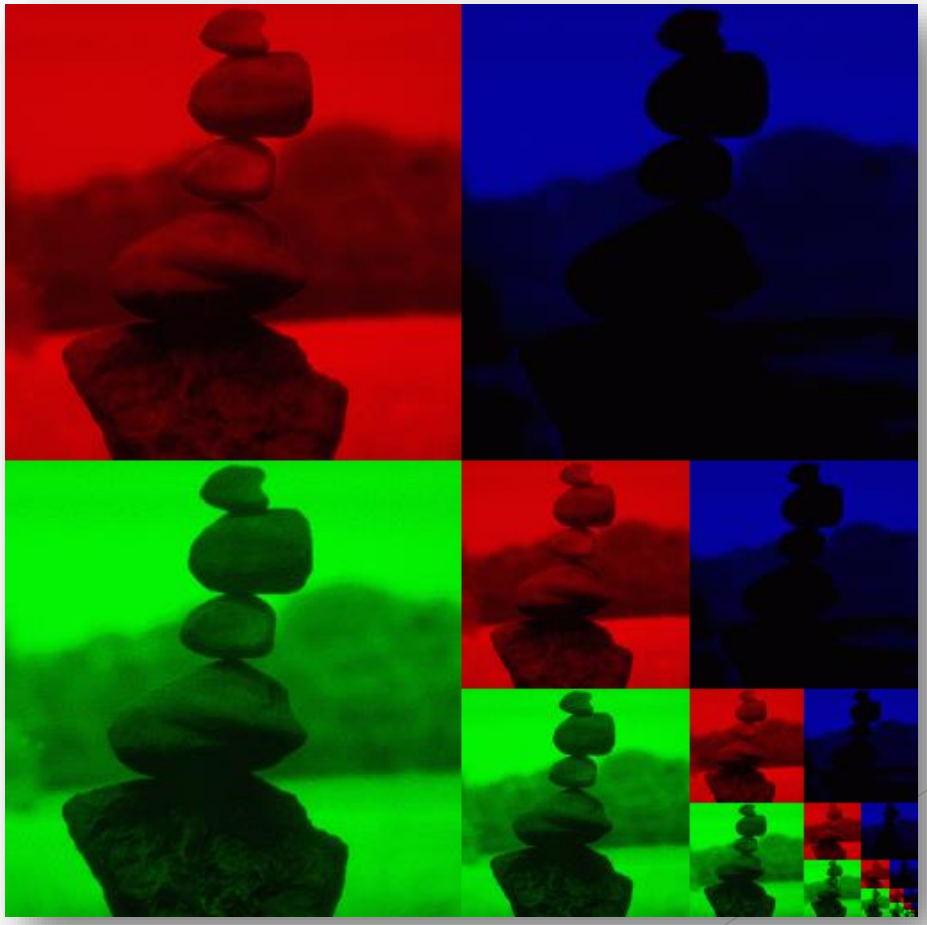
Zonder mipmap: aliasing



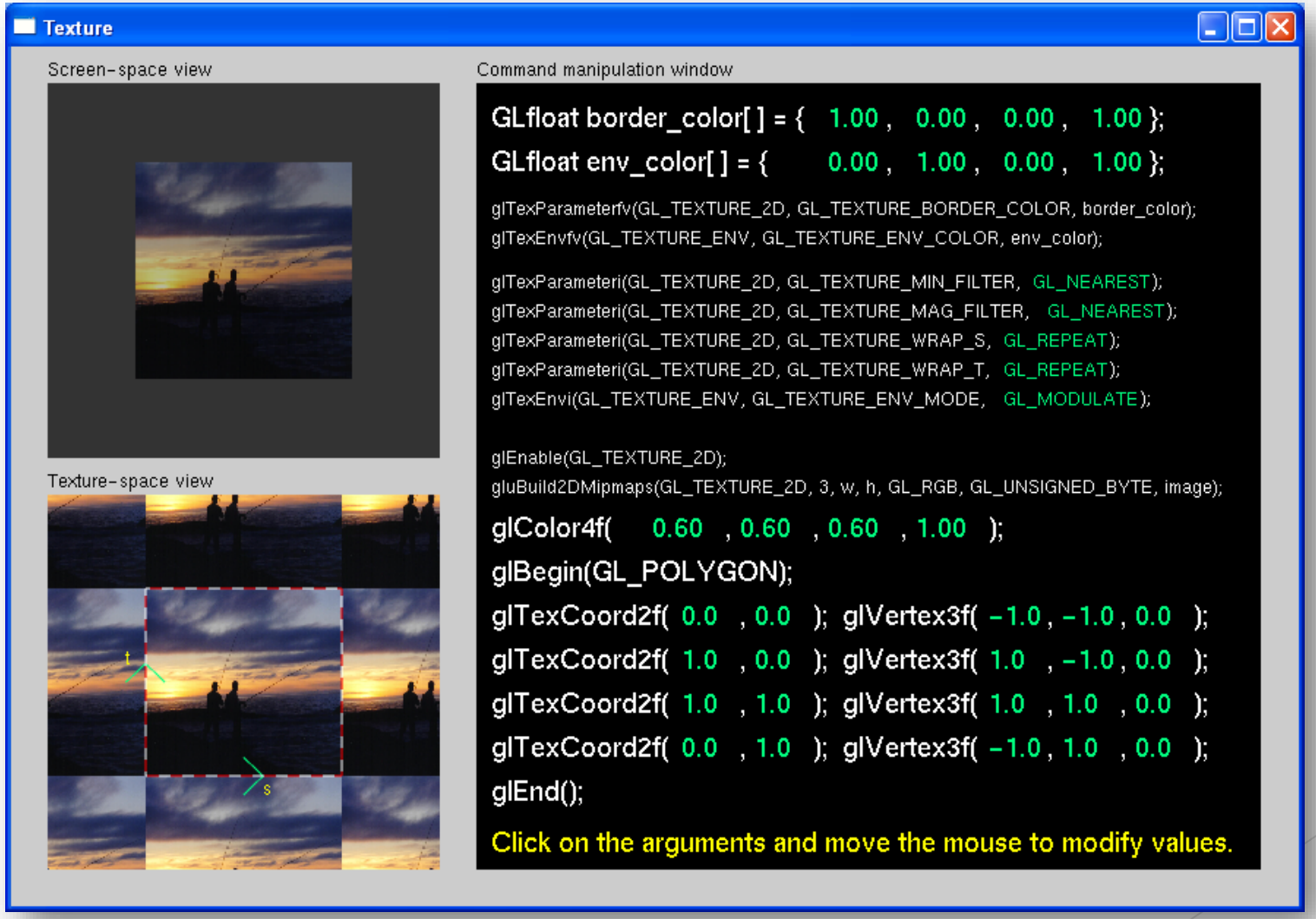
Met mipmap: minder aliasing

MIPmap opslag

Slechts 1/3 meer ruimte nodig!



GLtutor: texture

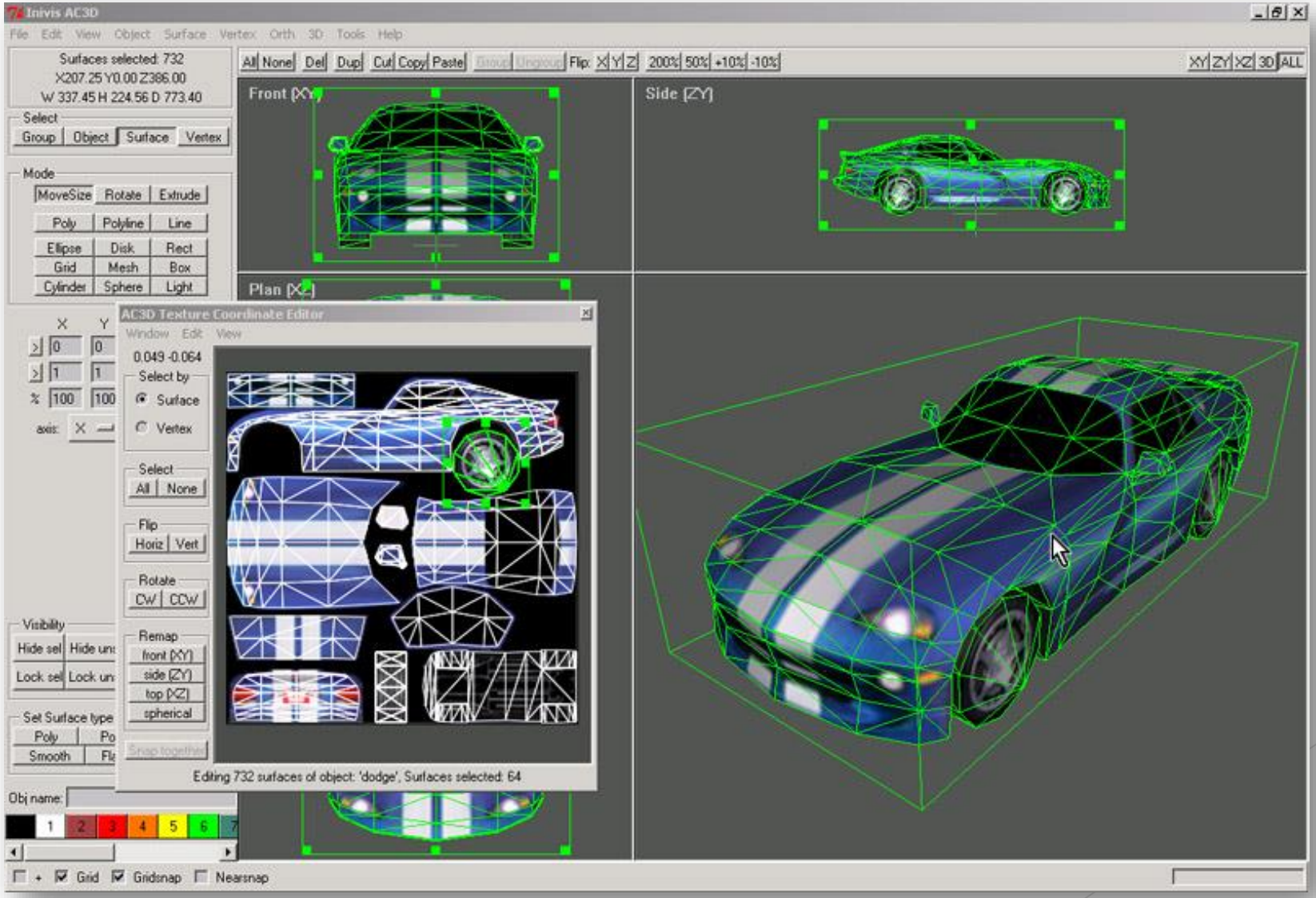


Texture coördinaat transformaties

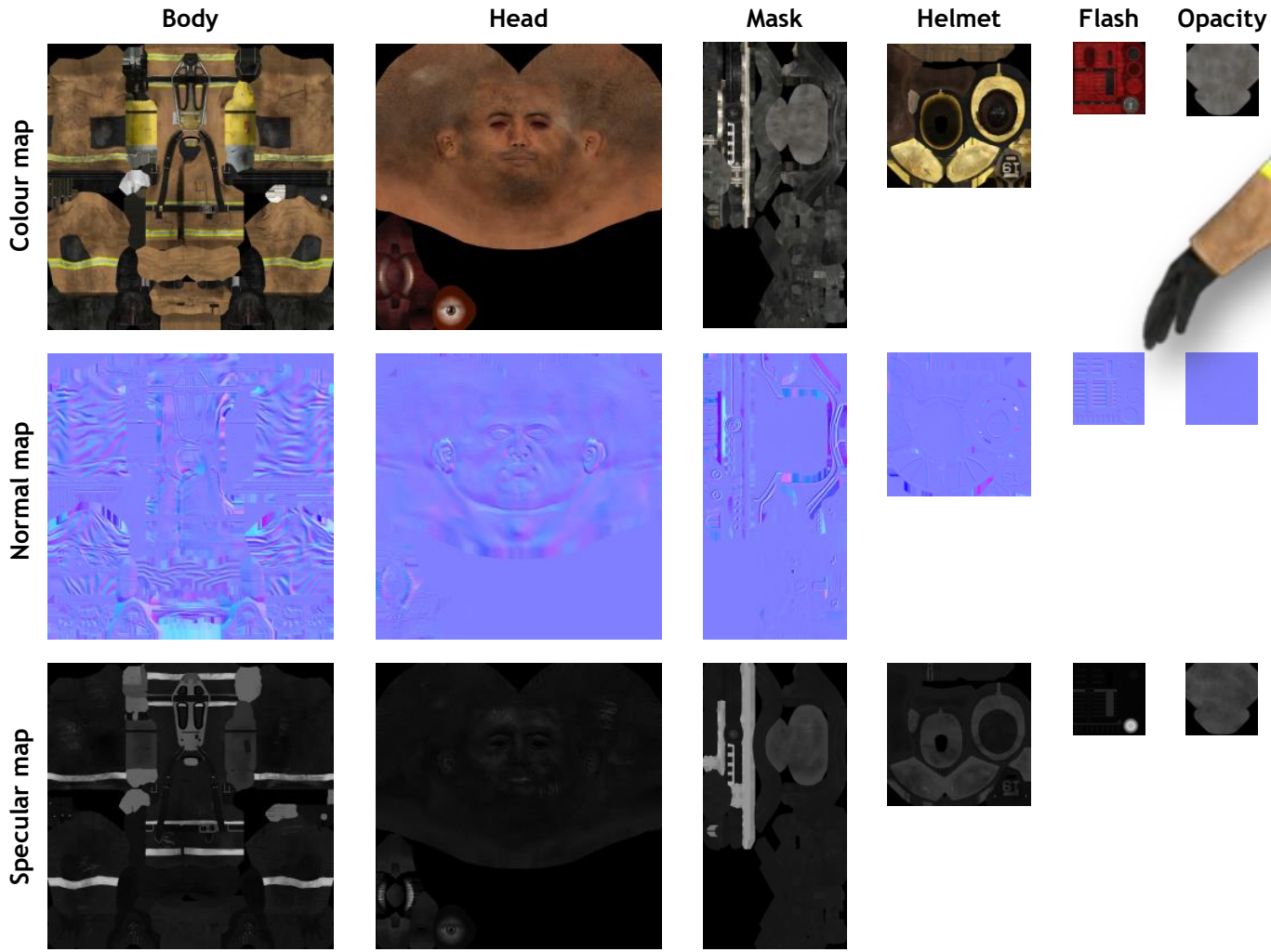
Verander positie, schaal en oriëntatie van een texture op een object

- Lineaire transformaties toegepast op texture coördinaten:
  - Translatie, rotatie, schaling, …
- Biedt meer flexibiliteit dan alleen (s, t) coördinaten

Texture coordinate editor



Example texture maps



Source: Microsoft RocketBox Avatar Library

Sphere Mapping

Polaire coördinaten voor een bol:

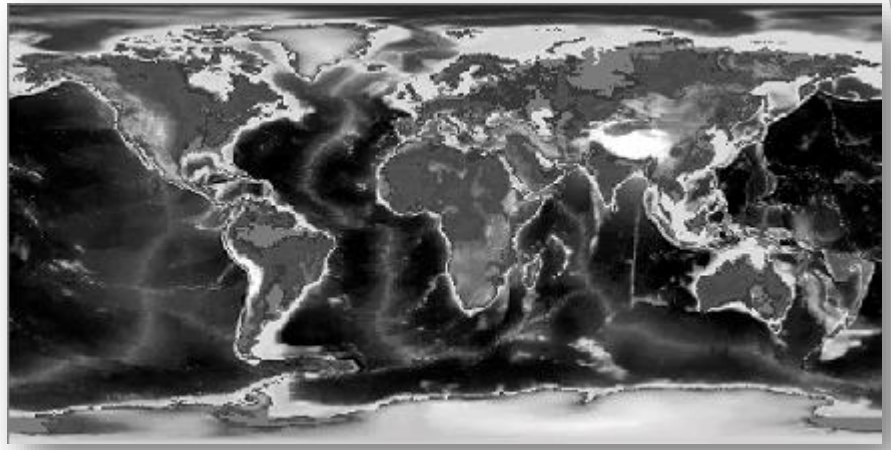
$$x = x_c + R\cos(\phi)\sin(\theta)$$
$$y = y_c + R\sin(\phi)\sin(\theta)$$
$$z = z_c + R\cos(\phi)$$

Dan zijn ϕ en θ:

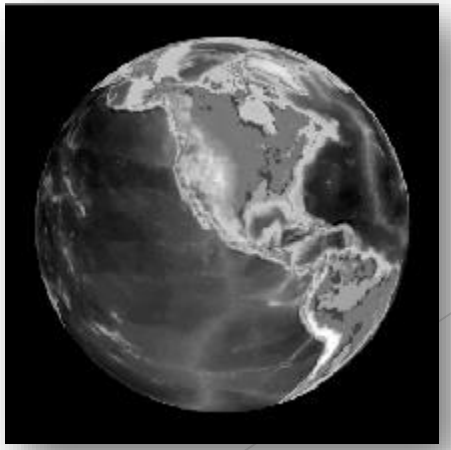
$$\phi = \arccos\left(\frac{z - z_c}{R}\right)$$
$$\theta = \arctan(y - y_c, x - x_c)$$

(ϕ, θ) in interval $[0, \pi] \times [-\pi, \pi]$

$$u = \frac{\phi}{2\pi}$$
$$v = \frac{\pi - \theta}{\pi}$$



Miller projectie



Toepassingen van texture mapping

1. Bump mapping
2. Displacement mapping
3. Shadow maps
4. Environment mapping

Toepassingen van textures

Behalve kleur is het ook mogelijk om andere materiaal of object eigenschappen te veranderen:

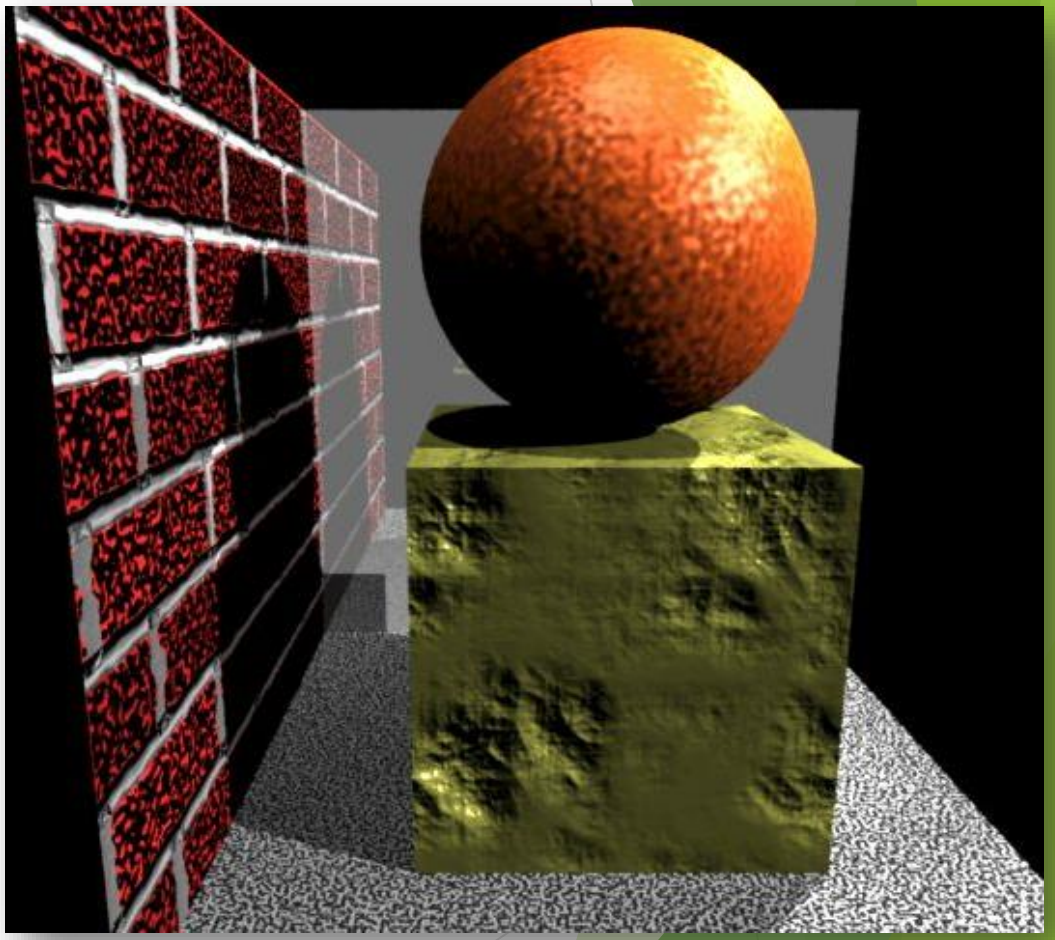
- surface normal (bump mapping)
- reflected color (environment mapping)
- schaduwen

1. Bump mapping

Het oppervlak van een object is vaak “ruw”

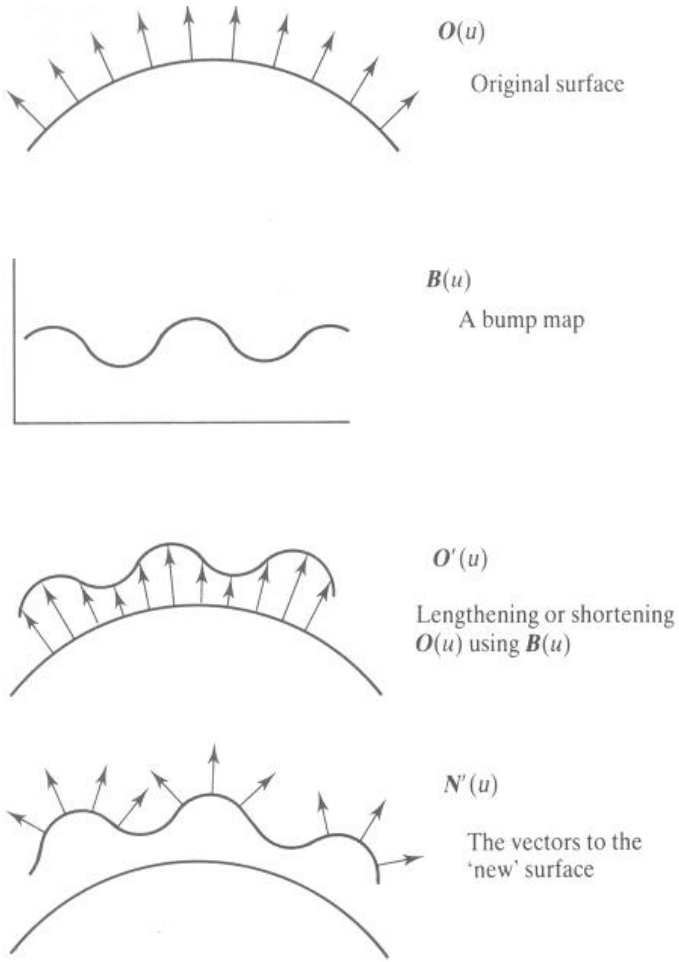
- Namaken in de vorm van geometrie is te complex

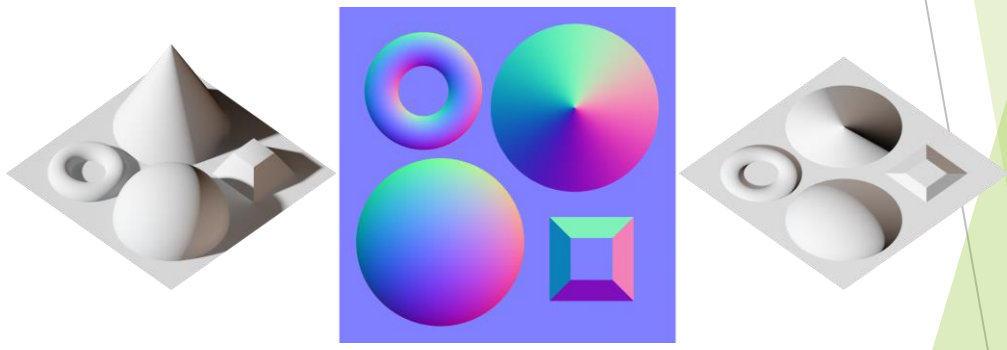
Door lokaal de normaalvector te beïnvloeden kan je dit effect ook bereiken



De “ruwheid” op deze objecten wordt veroorzaakt door “bump mapping”

1. Bump mapping





Example of a normal map (center) with the scene it was calculated from (left) and the result when applied to a flat surface (right). This map is encoded in tangent space. (Source: Wikipedia)

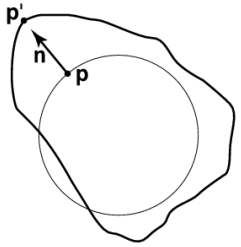
2. Displacement mapping

Bump mapping werkt niet goed als je kijkt naar het silhouette

- Schaduwen zijn ook vaak verkeerd

Oplossing: verander de oppervlaktestructuur

- Vereist “geometry shaders”
- Moet mogelijk zijn om het oppervlak op te delen



Bump mapped

Displacement mapped

3. Shadow maps

Schaduwen zijn makkelijk te berekenen met ray tracing. Maar hoe doe je dat in real-time apps?

- Vooraf uitrekenen (“baken”) naar shadow map
- Oppervlakken texturen met shadow maps

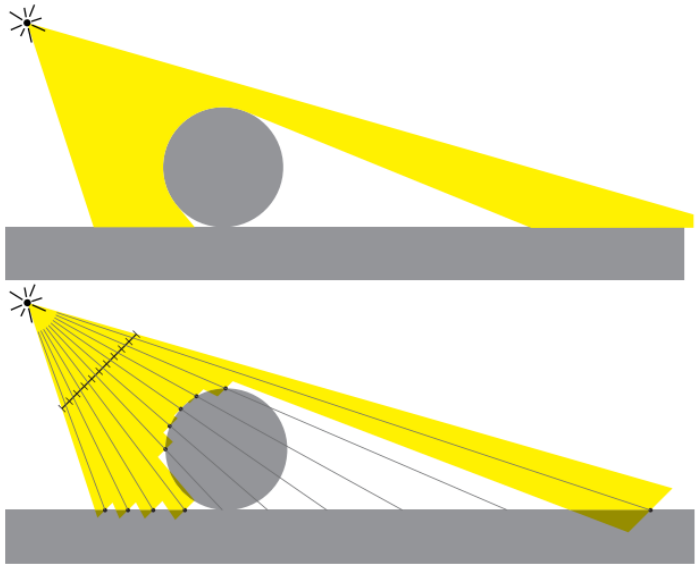
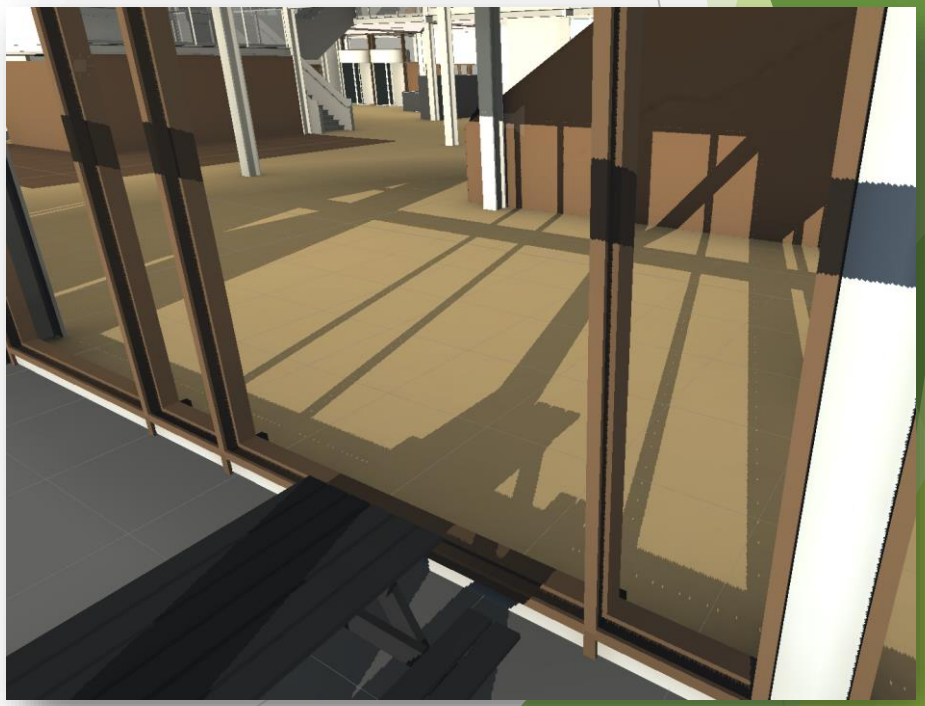


Figure 11.24. Top: the region of space illuminated by a point light. Bottom: that region as approximated by a 10-pixel-wide shadow map.



Shadow mapping in gebouw Lab42 gegenereerd met Unity

4. Environment mapping

Manier om omgeving in objecten te reflecteren:

1. Maak een texture van de omgeving: environment map
2. Texture object met environment map

Ook wel “reflection mapping” genoemd

4. Environment mapping

Wordt gebruikt om een object te modeleren dat de omgeving in het oog reflecteert

- Voor het eerst gebruikt in cyborg in “Terminator 2” (1991)

Verschillende benaderingen:

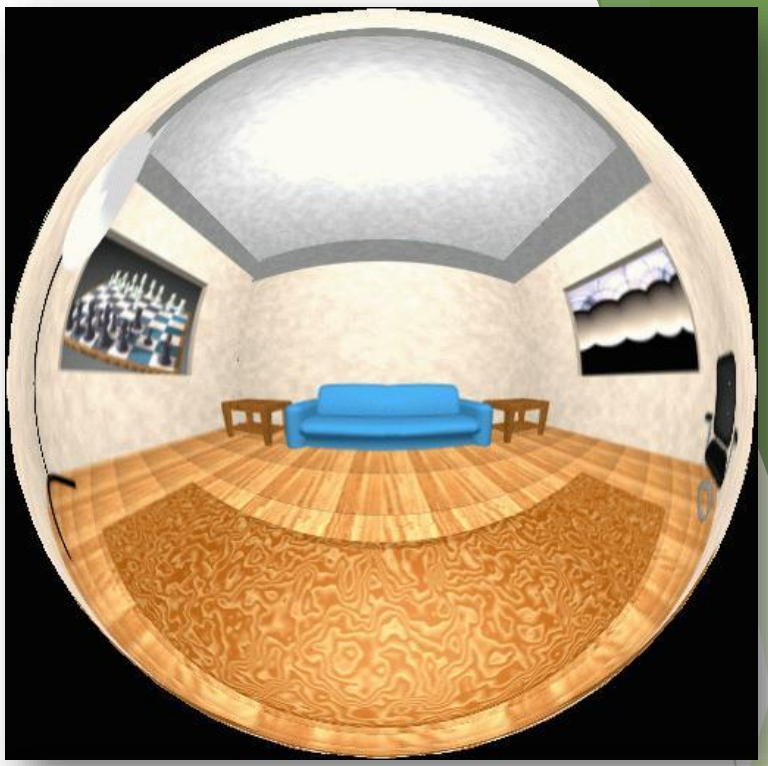
- Bol (“sphere map”)
- Kubus (“cube map”)

Cyborg van vloeibaar metaal in “Terminator 2: Judgement Day” (1991)

4. Environment mapping
Sphere map

Texture is een “fish-eye” aanzicht

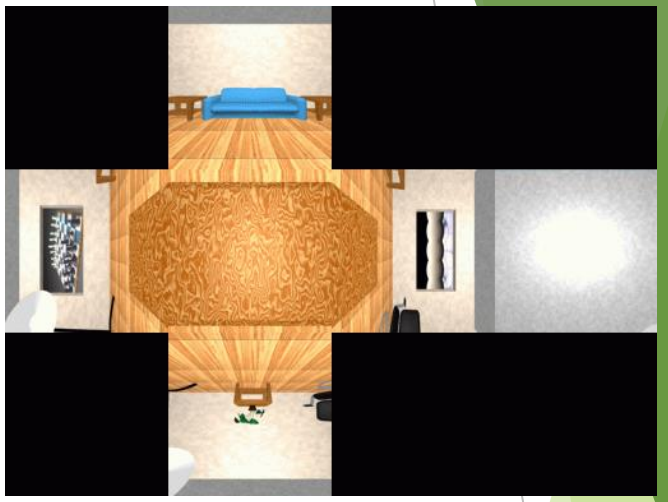
- Richt de camera naar een “spiegelbol”
- Spherische texture mapping maakt texture coördinaten die correct indiceren in de texture map



4. Environment mapping
Cube map

Zes 2D textures: zijden van een oneindig grote kubus

- Richt camera in 6 verschillende richtingen, weg van de oorsprong



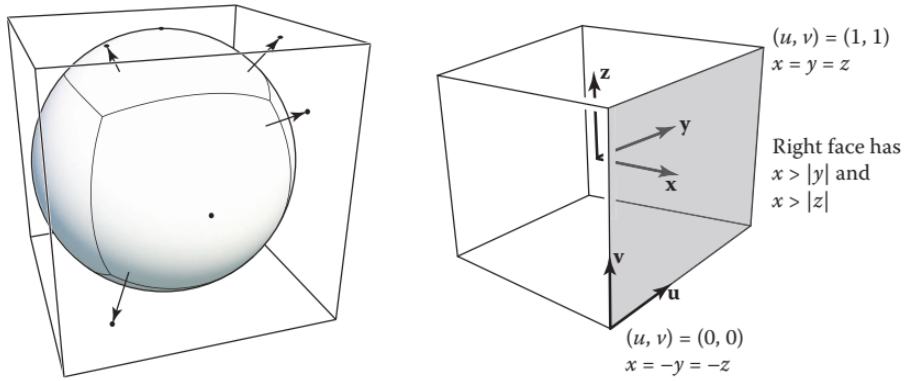
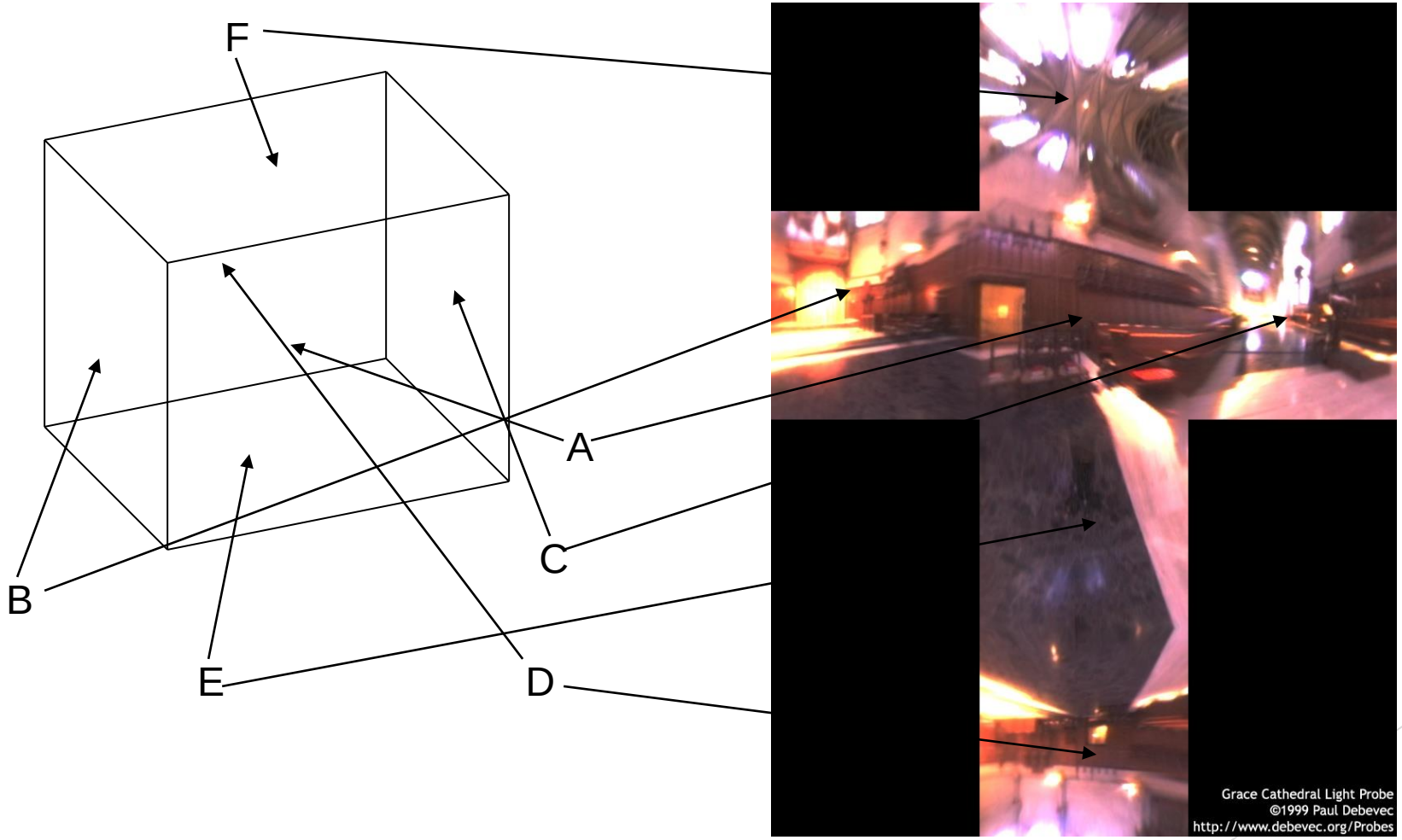


Figure 11.10. A surface being projected into a cubemap. Points on the surface project outward from the center, each mapping to a point on one of the six faces.

4. Environment mapping

Cube map



4. Environment mapping
Cube map

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 $r = (-0.2, 0.5, -0.84)$ selecteert de $-Z$ zijde
- De overige twee coördinaten, 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