PG430 – Innføring i grafikkprogrammering

Forelesning 7

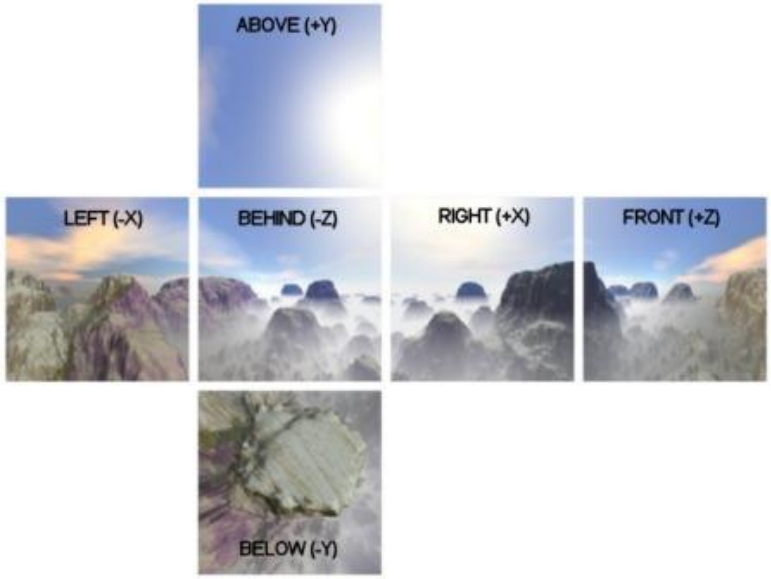
André R. Brodtkorb
Andre.Brodtkorb@nith.no

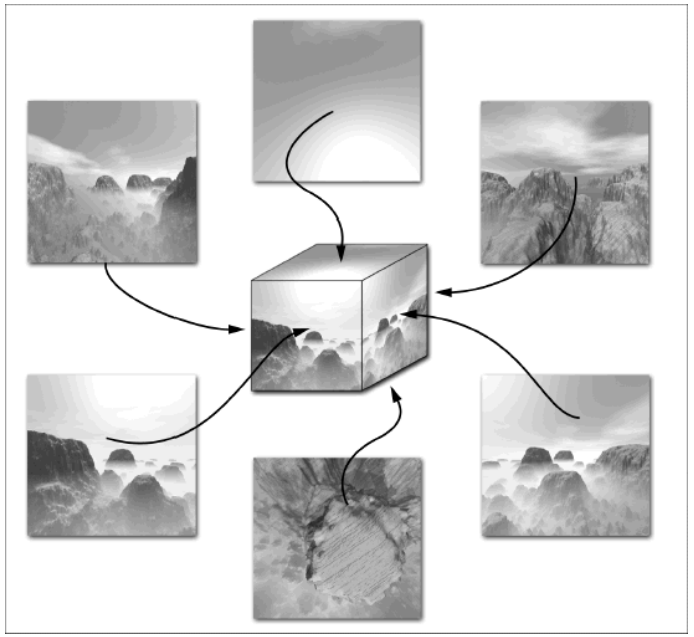
Oversikt

- Repetisjon fra forrige time
- Selection og feedback
- Fog
- Polygon offset
- OpenGL extensions
- Effektiv rendering av arrays
- Framebufferet: stencil buffer, accumulation buffer, etc
- Lab: Hjelp mappe 2

REPETISJON

Cube Maps

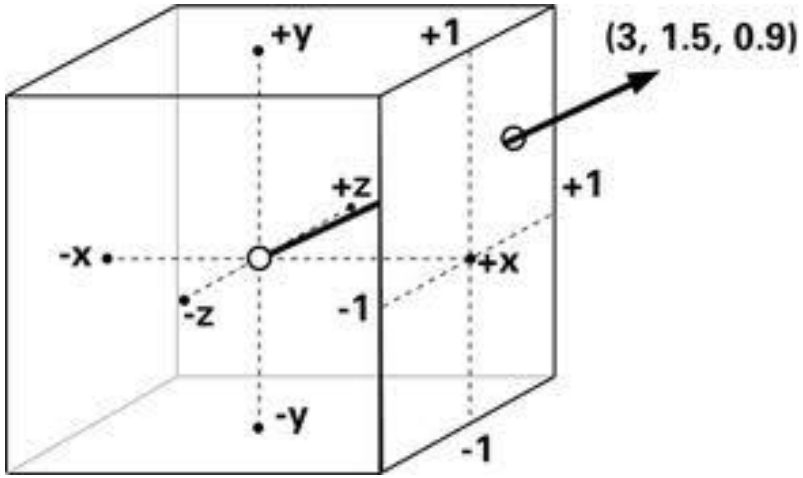




- glBindTexture(GL_TEXTURE_CUBE_MAP,TextureID)
  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGBA, .. Width, Height, 0, GL_RGBA, GL_UNSIGNED_BYTE, Data1)

Cube Map Texture Coordinates

- Cube maps use 3D texture coordinates!
- The vector points to one of the six faces
  - Easy to find: the largest component of the vector (x=3).
- Then find the texel by finding the 2D coordinate where the vector intersects with the face
  - Just find the point on the face:
    p = (3, 1.5, 0.9) / 3
    = (1, 0.5, 0.3) => (0.5, 0.3)



Shadow maps

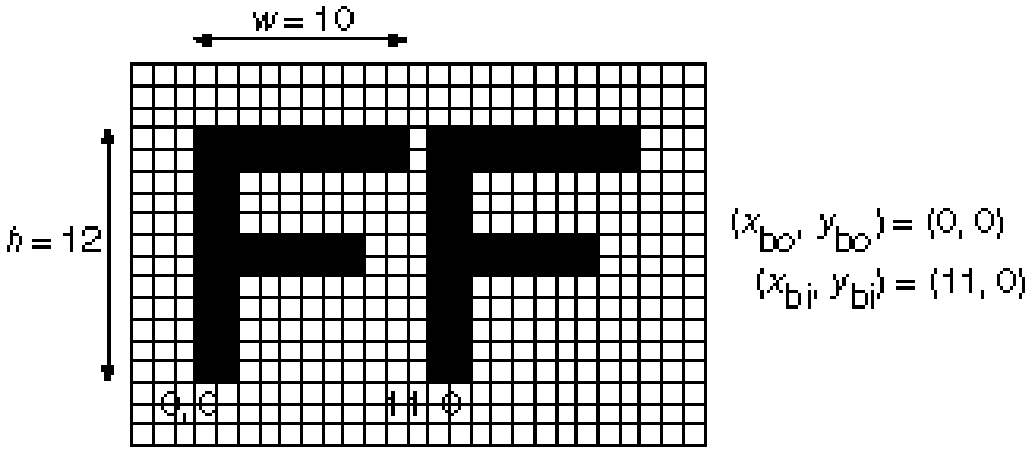
- På samme måte som vi kan bruke light maps til å beregne lyssetting kan vi også bruke shadow maps til å få naturlige skygger i scenen

Billboarding / Imposters

- Geometry is expensive to draw!
- We can use flat textures as "imposters"
- Examples:
  - Trees
  - Clouds
  - Asteroids

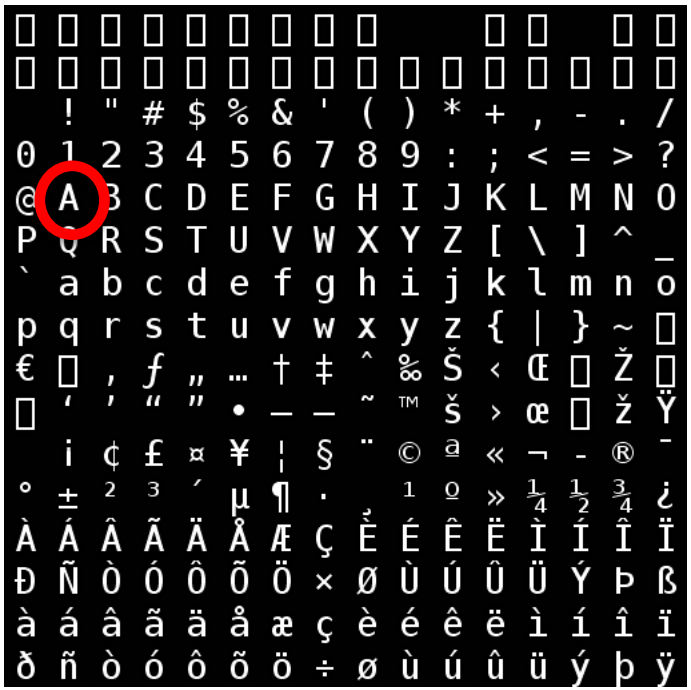
Tegning av bitmap forts.

glBitmap(10, 12, 0, 0, 11, 0, bitmap);



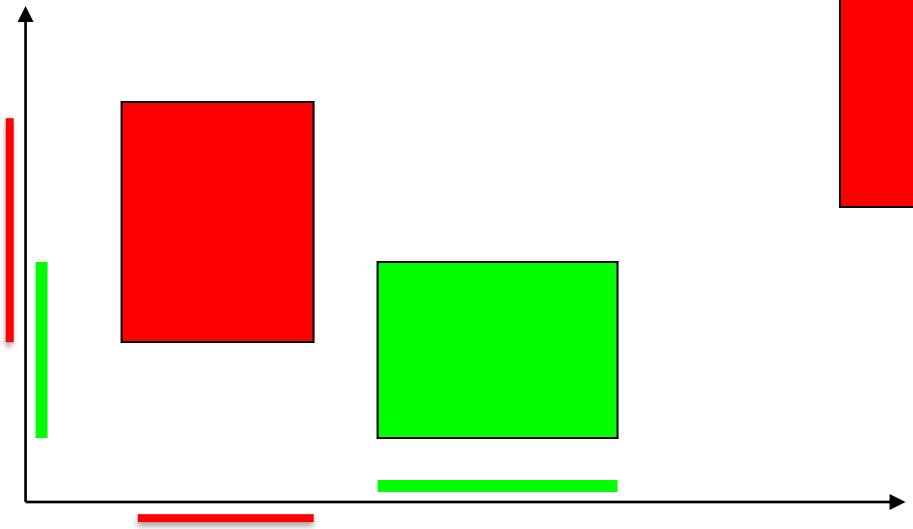
Teksturbaserte fonter

- Bruker quads og tekstur-koordinater til å skrive tekst
- Algoritme:
  - Lag en tekstur med ascii-symboler
  - Gjør om bokstav til nummer i ascii-tabellen (e.g., A=65)
  - Finn bokstaven i ascii-"matrisen":
    x = 65 % bredde = 1
    y = floor(65 / bredde) = 4
  - Gjør om x og y til teksturkoordinater
    s = x*font_bredde
    t = y*font_høyde



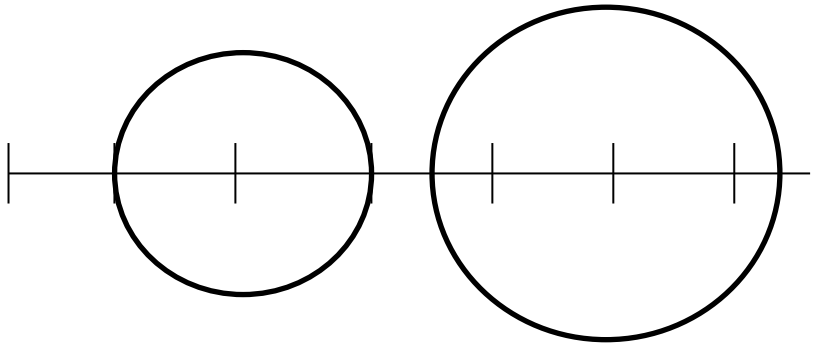
Kollisjonsdeteksjon: boks – boks

- To objekter har potensielt kollidert hvis deres bounding boxer har skjært hverandre.



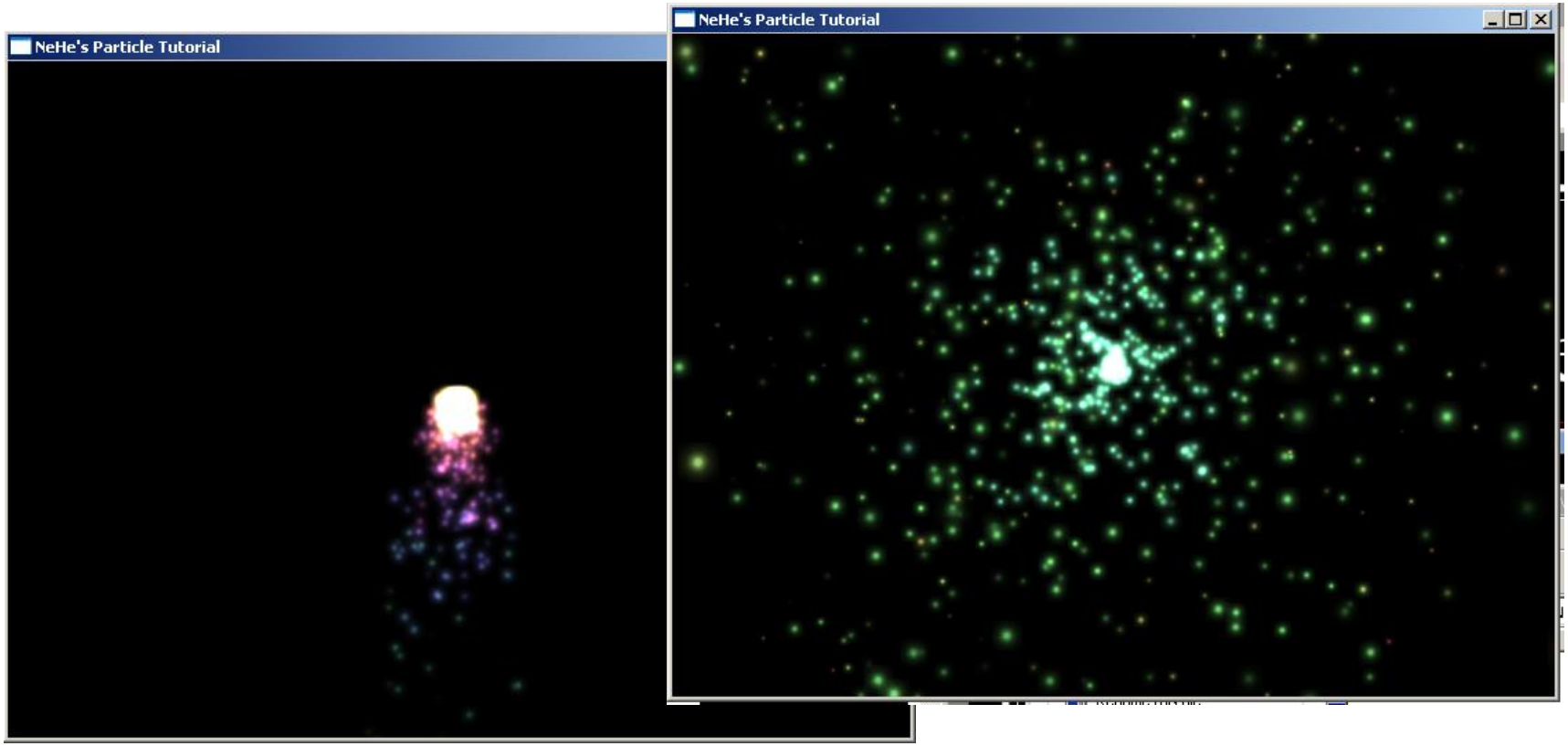
Kollisjonsdeteksjon: kule – kule

- Kodeeksempel:



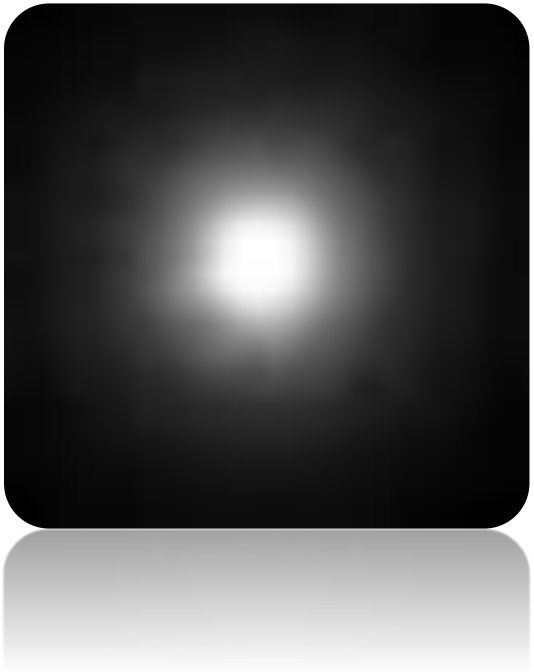
```
Vector3f a, b;
float a_radius = 1.0f;
float b_radius = 1.5f;

a = Vector3f(2.0, 0.0, 0.0);
b = Vector3f(5.0, 0.0, 0.0);

float dist = (a-b).length();
if(dist < a_radius + b_radius) std::cout << "Kollisjon";
else std::cout << "Ikke kollisjon";
```

Partikkeleffekter



Definisjonen av en partikkel

```
struct Particle {
Bool active;// Active (Yes/No)
Float life;// Particle Life
Float fade;// Fade Speed
Float r, g, b; //Particle color
Float x, y, z;// Position
Float xi, yi, zi; // Speed vector
}
Particle particles[max_particles];// Array of particles
```



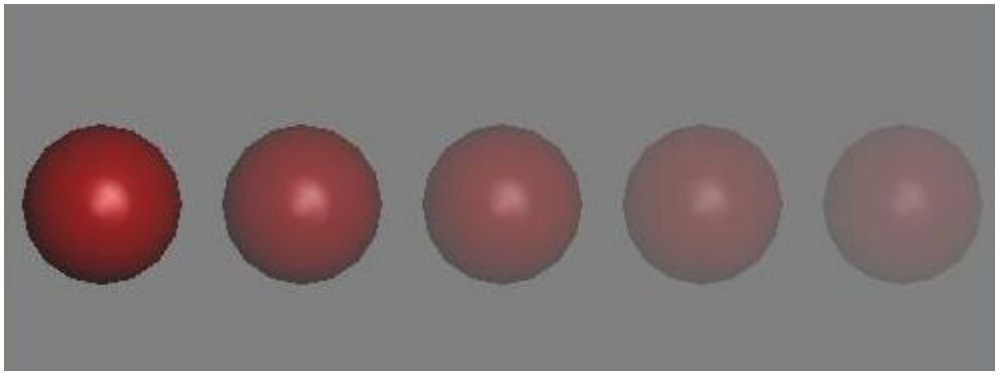
Andre partikkel-effekter

- Gravitasjon er lett å implementere!
  - Sett y-akselleras jon (ikke bare hastighet) til (9.81m/s^2)
- Selvlysende bier er lette å implementere!
  - Sett random retning hver update
- Stjernesimulering
  - Sett z-hastighet til Warp 3 og x- og y-hastighet til 0
- Brann, thrusters, etc

DAGENS FORELESNING

Fog - tåke

- *Fog* kan brukes til å simulere for eksempel tåke, røyk eller forurensning.
- Når man aktiverer *fog* vil objekter som er langt unna kamera gradvis forsvinne, og man vil til slutt kun se *fog*-fargen.
- Kan brukes til *culling* av objekter i en scene for å øke ytelsen.
- Gjør effekten av *popping* når objekter kommer innenfor *far plane* av viewport mindre (for eksempel partikler langt unna)



Likninger for beregning av *fog*

- GL_EXP:

$$f = e^{-(denisty \cdot z)}$$

- GL_EXP2:

$$f = e^{-(denisty \cdot z)^2}$$

- GL_LINEAR:

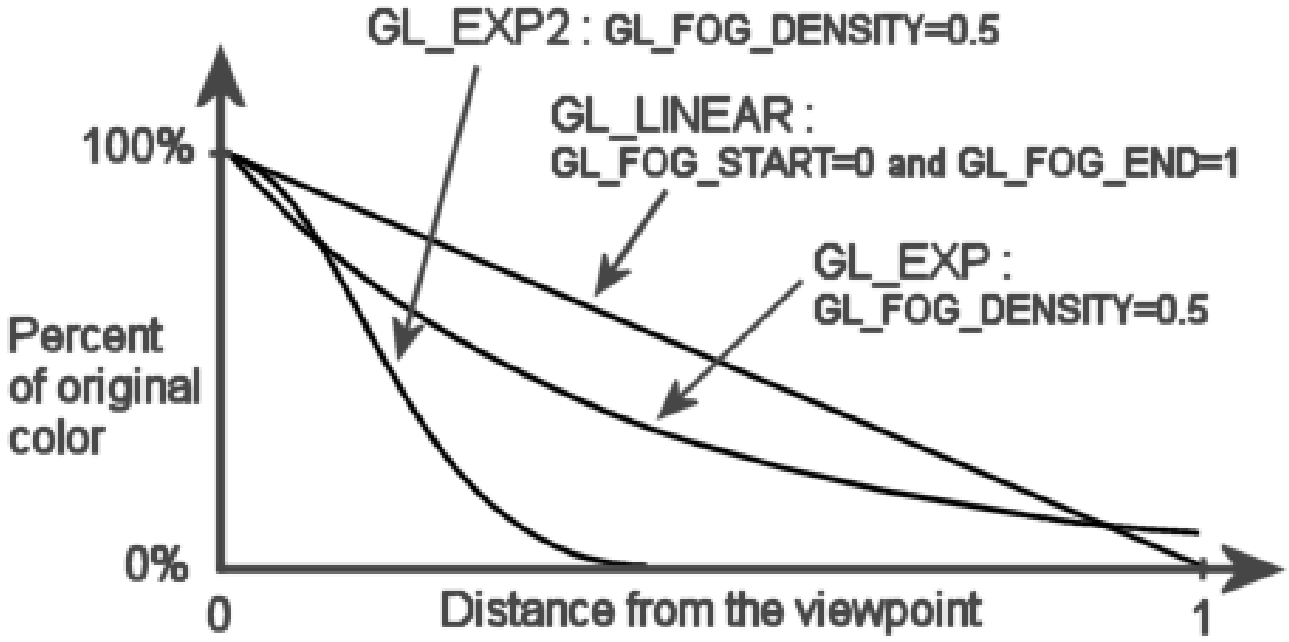
$$f = \frac{end - z}{end - start}$$

- Beregning av endelig farge:

$$C = f \cdot C_i + (1 - f) \cdot C_f$$

- Fog-farge blir blandet med fragment-farge ved å bruke en faktor *f*.
- Faktoren blir beregnet med en av tre likninger.
- *z* er avstanden mellom kamera og piksel
- *density* er tettheten
- *start* og *end* er dybdeverdier som angir start og stopp for beregning av *fog.*
- C_i er fragmentet sin farge mens C_f er fog-fargen.

Likninger for beregning av *fog*, forts.



Fog - kommandoer

- For å aktivere *fog* bruker man kommandoen:

```
glEnable(GL_FOG);
```

- Hvilken likning som skal brukes spesifiseres ved:

```
glFogi(GL_FOG_MODE, TYPE param);
```

param: GL_EXP, GL_EXP2, GL_LINEAR

- For å sette parametrene til likningene bruker man:

```
glFogf(GLenum pname, TYPE param);
```

pname: GL_FOG_DENSITY, GL_FOG_START, GL_FOG_END

- For å spesifisere fog-farge bruker man:

```
glFogfv(GL_FOG_COLOR, TYPE *params);
```

params: peker til fire RGBA fargeverdier.

Eksempel: Fog

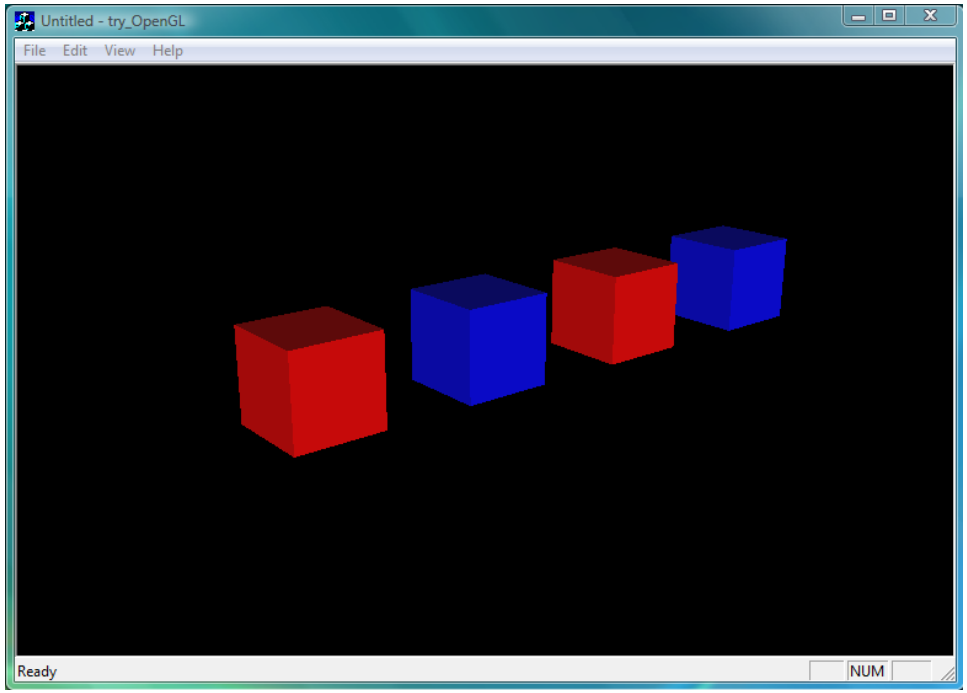
```
// aktiver fog
glEnable(GL_FOG);

// spesifiser hvilken likning som skal brukes
glFogi(GL_FOG_MODE, GL_LINEAR);

// spesifiser parametrene til likningen som brukes for å beregne fog
glFogf(GL_FOG_START, 1.0f);
glFogf(GL_FOG_END, 10.0f);

// per-pixel beregninger av fog
glHint(GL_FOG_HINT, GL_NICEST);
```

Demo: Fog

- Last ned Nate Robins fog demo fra “It’s Learning”
- Prøv å forandre på parametrene til de forskjellige likningene som brukes til å beregne fog.
- Observer forskjellen i tettheten på tåken mellom de forskjellige likningene.

Object selection

- Hvordan ”velge” objekter i scenen?
- Ray-casting/shooting
- Rendering primitives with unique colors
- **Selection buffer**



Object Selection

- *Selection bufferet* brukes hovedsakelig til å velge objekter i en scene ved hjelp av muspekeren.

- Mekanisme:
  - Tegn scenen til framebufferet.
  - Sett render modus til *select* og tegn scenen på nytt.

- Når man aktiverer *selection mode* kan man ikke tegne til framebufferet igjen før *selection mode* blir deaktivert.

Object Selection forts.

- For å angi hvilken måte man skal rendre på bruker man kommandoen:

GLint glRenderMode(GLenum mode)

mode: GL_RENDER, GL_SELECT, GL_FEEDBACK

Returverdi er antall *selection* treff; gir kun mening hvis modus er satt til GL_SELECT eller GL_FEEDBACK

- For å spesifisere hvilket array som skal brukes for å returnere selection data bruker man kommandoen:

glSelectBuffer(GLsizei size, GLuint *buffer)

size: maksimum antall verdier som kan lagres i arrayet.

buffer: peker til et array av unsigned integers hvor data lagres.

Man må kalle glSelectBuffer() før man aktiverer *selection* modus.

Name Stack

- Man kan spesifisere et navn per primitiv.
  - Flere primitiver kan ha samme ”navn” – brukes for å gi navn til objekter
  - Truffede navn returneres til select bufferet.
  - Ligner på glPushMatrix() og glPopMatrix().
- glInitNames()
  - Sletter navne-stacken slik at det er tom.
- glPushName(GLuint name)
  - Pusher *name* på navne-stacken.
- glPopName()
  - Poper ett navn fra toppen av navne-stacken.
- glLoadName(GLuint name)
  - Erstatter den øverste verdien i navne-stacken med *name.*

Steg ved bruk av selection

1. Spesifisere array som skal brukes til å returnere antall treff med glSelectBuffer()
2. Aktiver *selection mode*: glRenderMode(GL_SELECTION);
3. Initialiser navne-stacken.
4. Definer *viewing volume* for *selection*. Dette er ofte ikke det samme som standard *viewing volume*.
5. Tegn primitiver og bruk kommandoer for å manipulere navne-stacken.
6. Deaktiver *selection mode* og prosesser de returnerte dataene.

Object Selection

- Eksempel: Snowman
  - Klikk på snømennene
  - Navnet på klikket snømann kommer opp i konsollet
  - snowman.cpp:167-199: void renderScene()
    - snowman.cpp:85-110: void startPicking()
      - Initialiser picking
    - render.cpp:104-126: void draw()
      - Tegning av geometri og manipulasjon av navnestack'en
    - snowman.cpp:144-156: void stopPicking()
      - Henting og prosessering av pick results

Selection buffer

- Selection bufferet i OpenGL er ineffektivt
  - Er egentlig ikke grafikk-funksjonalitet
  - Implementert på CPU'en
    - Ved rendering av GPU-buffere må driveren overføre alle vertexer til CPU før selection!
- Hvordan gjøre det i moderne OpenGL?
  - Bruk farger istedenfor name stack!

Selection med farger

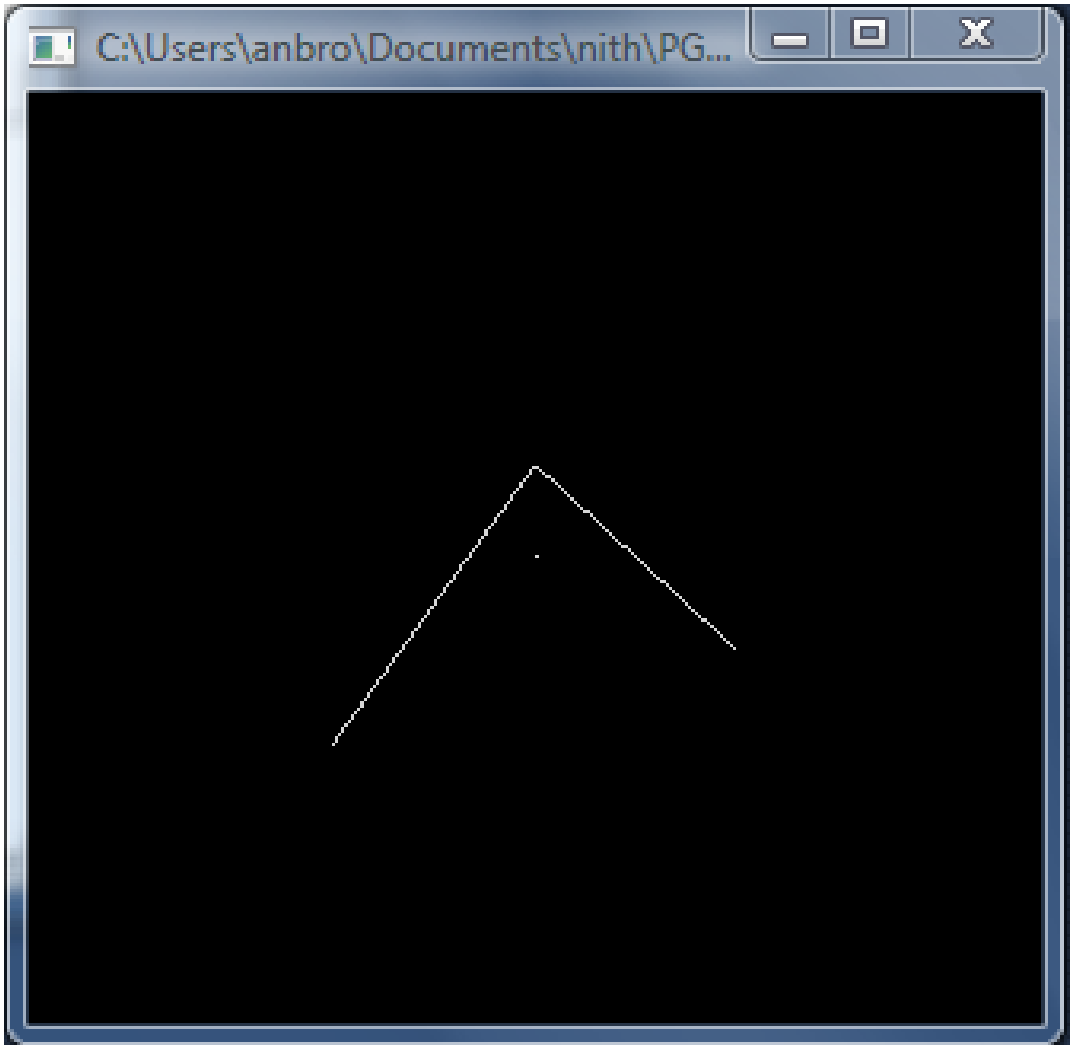
- Ha et eget render-pass som rendrer hvert objekt med en unik farge
  - Bruk en liten viewport, som kun dekker området du er interessert i.
  - Veldig enkelt prinsipielt, men krever at du fjerner teksturering, etc. fra "picking" render passet
- Bruk så glReadPixels til å lese tilbake til CPU

Feedback

- Veldig lik *selection mode*.
- I *feedback mode* blir informasjon om transformerte primitiver sendt tilbake som et array av flyttallsverdier.
- Spesifiserer type primitiv (punkt, linje, triangel osv.) etterfulgt av per-vertex-data (vertex pos., farge osv.)
- *Feedback mode* aktiveres med funksjonskallet glRenderMode() og GL_FEEDBACK som argument.

Feedback

- Hva blir output fra feedback?



Feedback - eksempel

```
// output består av [x y z r g b a]
GL_LINE_RESET_TOKEN
  30.00 30.00 0.00 0.84 0.84 0.84 1.00
  50.00 60.00 0.00 0.84 0.84 0.84 1.00
GL_LINE_TOKEN
  50.00 60.00 0.00 0.84 0.84 0.84 1.00
  70.00 40.00 0.00 0.84 0.84 0.84 1.00
GL_PASS_THROUGH_TOKEN
  1.00
GL_PASS_THROUGH_TOKEN
  2.00
GL_POINT_TOKEN
  50.00 50.00 0.00 0.84 0.84 0.84 1.00
```

```
glBegin (GL_LINE_STRIP);
  glNormal3f (0.0, 0.0, 1.0);
  glVertex3f (30.0, 30.0, 0.0);
  glVertex3f (50.0, 60.0, 0.0);
  glVertex3f (70.0, 40.0, 0.0);
glEnd ();
if (mode == GL_FEEDBACK)
  glPassThrough (1.0);
glBegin (GL_POINTS);
  glVertex3f (-100.0, -100.0, -100.0);  /*  will be clipped  */
glEnd ();
if (mode == GL_FEEDBACK)
  glPassThrough (2.0);
glBegin (GL_POINTS);
  glNormal3f (0.0, 0.0, 1.0);
  glVertex3f (50.0, 50.0, 0.0);
glEnd ();
```

Feedback

- Som med selection er feedback også tregt
- Bør kun brukes for debugging ol.

OpenGL Extensions

- OpenGL blir spesifisert av en gruppe av som kalles Khronos.
- Khronos fokuserer på åpne standarder, og består av alle de ledende aktørene innen fagfeltet bortsett fra Microsoft.
- Hver produsent kan spesifisere tilleggsfunksjonalitet til OpenGL. Dette kalles OpenGL extensions.
- Før man bruker utvidelser bør man sjekke om driveren støtter utvidelsene
- Noen utvidelser blir en del av senere OpenGL versjoner.

OpenGL Extensions og Windows

- Default støttes ikke annet enn OpenGL versjon 1.1 fra 1996 i Windows.

- Sjekk **gl.h** og du finner dette

```
#define GL_VERSION_1_1
```

- Vi har kun brukt denne gamle funksjonaliteten så langt i kurset.

- For å ta i bruk nyere versjoner av OpenGL for Windows må man bruke extensions.

OpenGL extensions under Windows

- Funksjonalitet nyere enn default støttet av Windows er litt komplisert å bruke (man må hente funksjonspekere).

- GLEW løser dette problemet på en platformuavhengig måte.

- The OpenGL Extension Wrangler Library (GLEW)
  - The latest release contains support for OpenGL 4.2
  - Må inkludere GL/glew.h istedenfor eller før GL/gl.h
  - Må kalle glewInit(); etter OpenGL er initialisert, før extensions brukes.

- Det finnes andre alternativer: GL Easy Extension library (Glee)
  - Supports OpenGL functions up to version 3.0, så bruk heller glew

GLEW eksempel

- Installeres på C:\libraries\i386\glew-1.7.0\
  - C:\libraries\i386\glew-1.7.0\include\GL\glew.h
- I starten av filen:
  #include <GL/glew.h>
- I init

```
init() {
    …
    Glenum err = glewInit();
    if (err != GLEW_OK) {
        …
    }
}
```

Effektiv rendering av arrays

Immediate mode (repetisjon)

- En kommando kan kjøres umiddelbart når den kalles. Dette kalles immediate mode.
- Kommandoer for å tegne geometri, forandre stater, sette transformasjonen osv. blir sendt en om gangen til OpenGL for rendering.
- Dette er den enkleste måten å rendre geometri på i OpenGL; det er denne metoden vi har brukt så langt.
- Dette regnes som den minst effektive metoden man kan bruke for rendering.

Immediate mode forts. (repetisjon)

- Typisk kode:

```
glBegin(GL_TRIANGLES);
    glNormal3f(0.0f, 0.0f, 1.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
glEnd();
```

Display-lister (repetisjon)

- En display-liste er en samling av OpenGL kommandoer som har blitt lagret for å kjøres senere.
- På noen typer grafikk-hardware kan display-lister bli lagret i dedikert minne.
- Ved bruk av display-lister kan data bli lagret på en optimalisert måte som er tilpasset den hardware som brukes.
- Kan ikke forandres når den er laget. Liten fleksibilitet.
- Godt egnet for å rendre samme geometri flere ganger:
  - Eksmpel bil: Lag geometrien for et hjul i en dislay-liste. Kall denne display-lista fire ganger med forskjellig modelview matrise.

Display-liste forts. (repetisjon)

- I funksjonen init()

```
list_id = glGenLists(1);                    // list_id er en integer

glNewList(list_id, GL_COMPILE);
   glBegin(GL_TRIANGLES);
      glNormal3f(0.0f, 0.0f, 1.0f);
      glVertex3f(0.0f, 0.0f, 0.0f);
      glVertex3f(1.0f, 0.0f, 0.0f);
      glVertex3f(1.0f, 1.0f, 0.0f);
   glEnd();
glEndList();
```

- I funksjonen render()

```
glCallList(list_id)
```

Vertex Arrays (repetisjon)

- Reduserer antall funksjonskall i forhold til display-lister og immediate mode.
- All vertex data lagres i arrays.
  - Vertex array, normal array, color array osv ...
- Vertexer som brukes av forskjellige primitiver må spesifiseres mer enn en gang.
- Moderne hardware er optimalisert med hensyn på vertex arrays.

Vertex Arrays forts. (repetisjon)

- For å sette posisjon, farge og normal må man:
- Enable vertex arrays:
  - glEnableClientState(GL_VERTEX_ARRAY);
  - glEnableClientState(GL_NORMAL_ARRAY);
  - glEnableClientState(GL_COLOR_ARRAY);
- Sette vertex arrays
  - glVertexPointer(...);
  - glNormalPointer(...);
  - glColorPointer(….);
- Disable vertex arrays:
  - glDisableClientState(...);

Vertex Arrays forts. (repetisjon)

```
// spesifiser vertex array
GLfloat vertices[ ] = {0,0,1,  0,0,1,  0,0,1,  0,0,1,
                       1,0,0,  1,0,0,  1,0,0, 1,0,0,
                         0,1,0,  0,1,0,  0,1,0, 0,1,0,
                         -1,0,0,  -1,0,0, -1,0,0,  -1,0,0,
                         0,-1,0,  0,-1,0,  0,-1,0,  0,-1,0,
                         0,0,-1,  0,0,-1,  0,0,-1,  0,0,-1   };
// render geometry
glEnableClientState(GL_VERTEX_ARRAY);    // enable vertex arrays
glVertexPointer(3, GL_FLOAT, 0, vertices);     // sette vertex pointer
glDrawArrays (GL_QUADS, 0, 24);                // tegne 6 quads
glDisableClientState(GL_VERTEX_ARRAY);  // disable vertex arrays
```

Mer kompakt format, slipper å sende mange kommandoer til OpenGL.
Bruker kun et funksjonskall på f.eks 10000 vertexer.

Indekserte arrays

- Samme antall funksjonskall som vanlige vertex arrays

- Vertexer som brukes av forskjellige primitiver må ikke spesifiseres mer enn en gang.
  - Merk: det brukes samme indeks til posisjon, farge, normal osv
  - **Reduserer ofte datamengden dramatisk!**

glDrawElements(GLenum *mode*, GLsizei *count*, GLenum *type*, const GLvoid **indices*) ;

Eks, tegner triangler utspent av 18 vertices:

```
glDrawElements(GL_TRIANGLES, 18 ,
 GL_UNSIGNED_INT, indicies);
```

Indekserte arrays (forts)

```
// spesifiser vertex array
GLfloat vertices[ ] = {0,0,0,  0,0,1,  0,0,2,  0,0,3,
            0,1,0,  0,1,1,  0,1,2,  0,1,3};

Gluint indecies[] = {0, 1, 4,  4, 1, 5,
          1, 2, 4,  5, 2, 6,
          2, 3, 5,  6, 3, 7};
// render geometry
glEnableClientState(GL_VERTEX_ARRAY);    // enable vertex arrays
glVertexPointer(3, GL_FLOAT, 0, vertices);     // sette vertex pointer
glDrawElements(GL_TRIANGLES, 18, GL_UNSIGNED_INT, indicies);

glDisableClientState(GL_VERTEX_ARRAY);  // disable vertex arrays
```

Mer kompakt format, slipper å sende mange kommandoer til OpenGL.

Bruker kun et funksjonskall på f.eks 10000 vertexer.

Vertex Buffer Object

- Vertex buffer object brukes for å la vertex arrays ligge i grafikk minne.
  - Veldig likt vertex arrays i tankegang
  - Men nå overføres "ingen" data mellom CPU og GPU for hvert renderkall
- Kan brukes både for både indekserte og ikke indekserte vertex arrays
- Kan også ”mappe” VBO til system minne for å oppdatere hele eller deler av en VBO

Viktige funksjonskall for VBO

- void glGenBuffers (GLsizei n, GLuint* ids)
  - Tilsvarende til glGenTextures
- void glBindBuffer (GLenum target, GLuint id)
  - Target er GL_ARRAY_BUFFER, eller GL_ELEMENT_ARRAY_BUFFER for indeks array
- void glBufferData (GLenum target, GLsizei size, const void* data, GLenum usage)
  - Size angir antall byte, usage er GL_STATIC_DRAW for statiske arrays. Usage brukes som hint til OpenGL for å fortelle hvordan VBOen vil bli brukt
- void glDeleteBuffers (GLsizei n, const GLuint* ids)

VBO – init()

```
void init() {
…
//Lag id for et buffer
glGenBuffers(1, &posVBO_);
//Alloker og bind bufferet
glBindBuffer(GL_ARRAY_BUFFER, posVBO_);
//Last opp data
glBufferData(GL_ARRAY_BUFFER, 4*posArray_.size(), &posArray_[0],
    GL_STATIC_DRAW);
//Unbind bufferet
glBindBuffer(GL_ARRAY_BUFFER, 0);
...
}
```

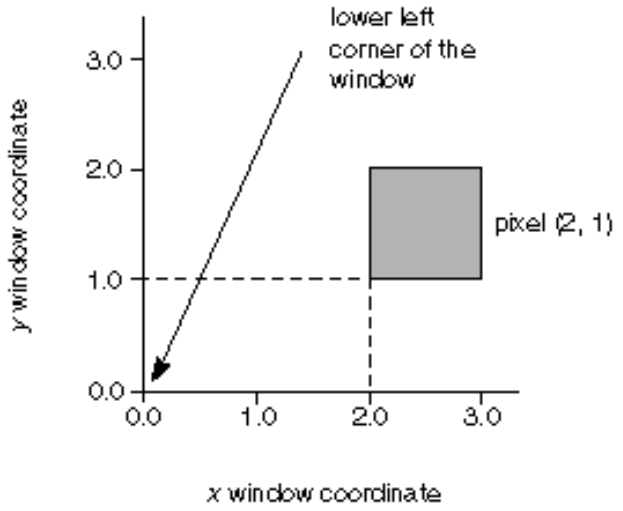
VBO – render()

```
void render() {
...
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, posVBO_);
//Legg merke til hva pekeren er satt til
glVertexPointer(3, GL_FLOAT, 0, (GLvoid*) 0);
glDrawArrays(GL_TRIANGLES, 0, 3);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glDisableClientState(GL_VERTEX_ARRAY);
...
}
```

Framebuffer

Framebuffer

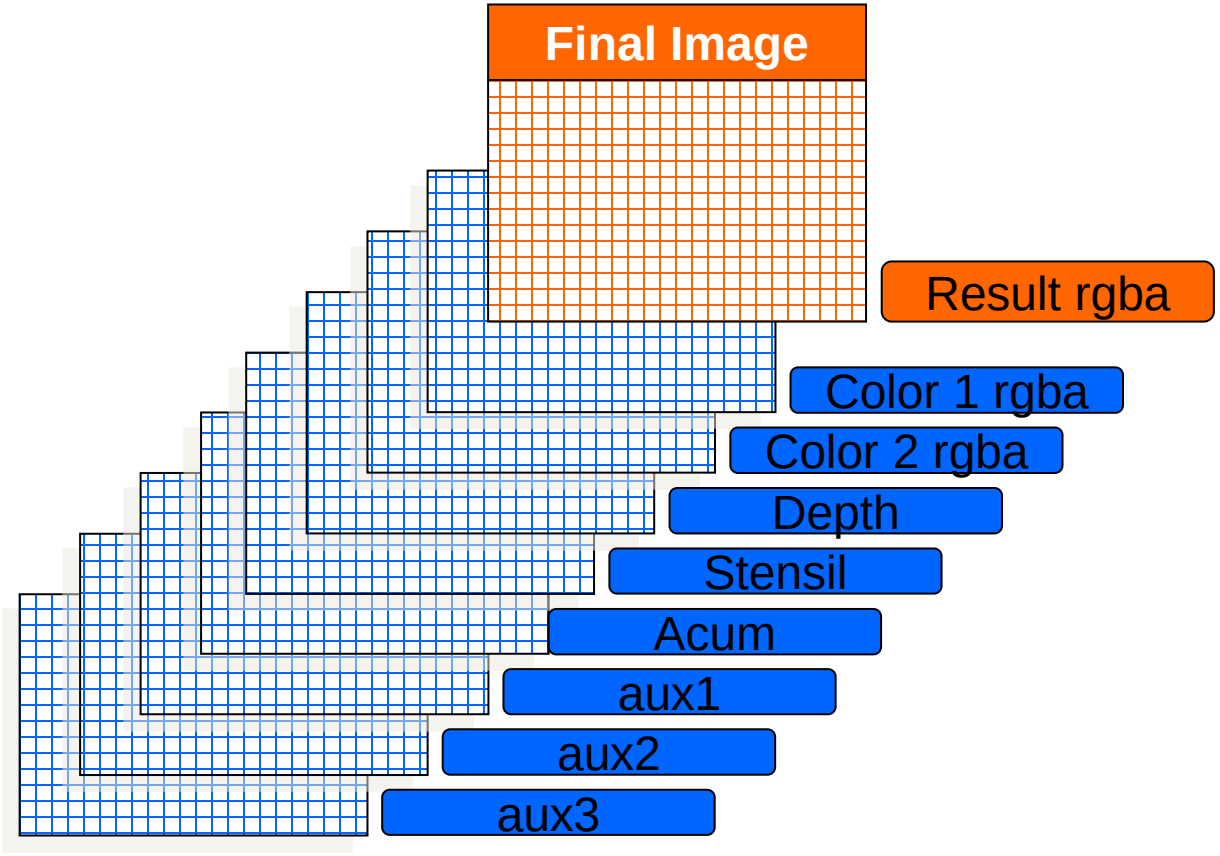
- Det siste steget i OpenGL pipelinen er framebufferet. Her blir fragmenter omgjort til piksler.
- Data som er lagret uniformt for hver piksel utgjør et buffer.
- Pikselen i nedre venstre hjørne i et OpenGL vindu er piksel (0,0).



Framebuffer forts.

- Et framebuffer inneholder informasjon om piksler.
- Et framebuffer kan bestå av flere forskjellige underbuffere:
  - Fargebuffere
  - Dybdebuffer
  - Stencil-buffer
  - Accumulation-buffer
- Alle disse bufferene må lages når OpenGL-konteksten opprettes (SDL-kommandoer)

Framebuffer forts.



Fargebuffer

- Fargebuffere inneholder fargeverdier (RGBA)
- Ofte endelige bildet som skal tegnes til skjermen.

Fargebuffer forts.

- For å angi hvilket fargebuffer man skal skrive til og lese fra:

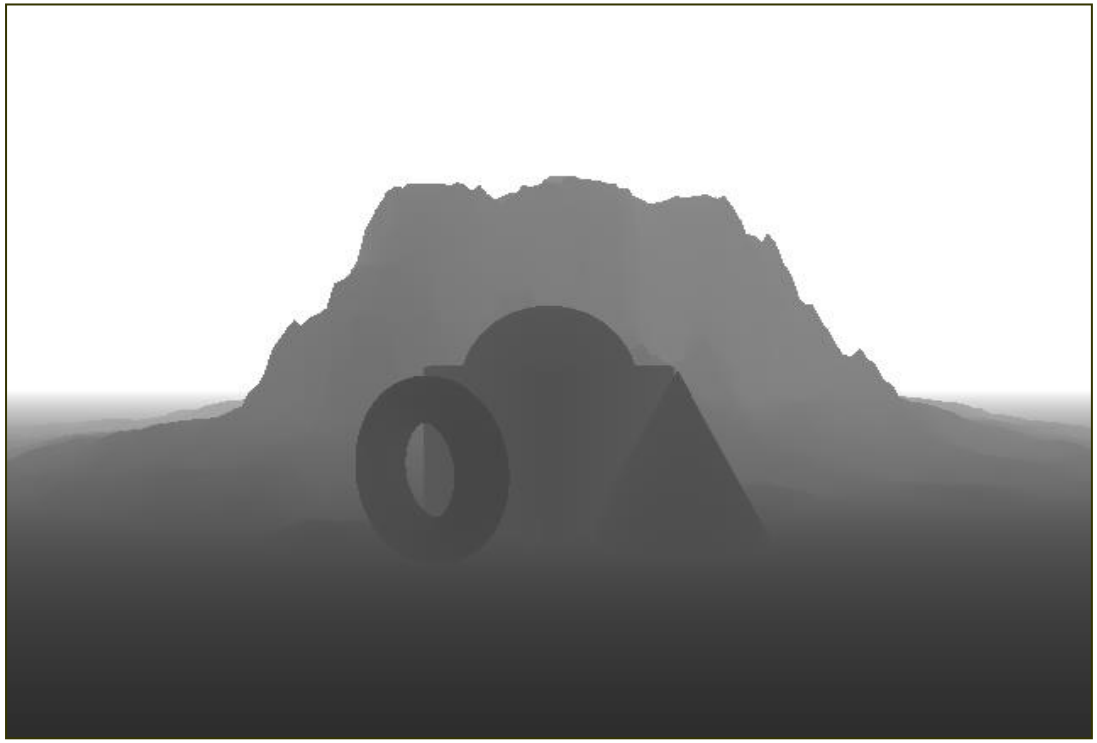
  glDrawBuffer(GLenum mode);

  glReadBuffer(GLenum mode);

  – mode: GL_FRONT, GL_BACK, GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT ...

- For å hindre oppdatering av enkelte farger bruk:
  – glColorMask(Glboolean red, GLboolean green, GLboolean blue, GLboolean alpha)
  – Eks: glColorMask(GL_TRUE, GL_FALSE, GL_FALSE, GL_FALSE)
    - Kun rødfrarge blir oppdatert

Dybdebufferet



- Inneholder dybdeverdiene til pikslene, avstand fra kamera

Dybdetest

- Brukes hovedsakelig for å teste om en piksel befinner seg nærmere kamera enn en piksel som er tegnet tidligere.
- Hvis pikselen er nærmere skal den tegnes, er den lengre unna skal den ikke tegnes.

- glEnable(GL_DEPTH_TEST);

- Dybdetest-funksjonaliteten kan forandres med:

glDepthFunc(GLenum func);

func: GL_NEVER, GL_ALWAYS, GL_LESS, GL_GREATER osv ... default GL_LESS

- Kan hindre oppdatering av dybdebuffer med: glDepthMask(GLboolean flag)

Alpha test

- Alpha test gir mulighet for å akseptere eller forkaste et fragment basert på alpha-verdi.

- glEnable(GL_ALPHA_TEST);

- Alpha-test-funksjonaliteten kan forandres med:

glAlphaFunc(GLenum func, GLclampf ref);

func: GL_NEVER, GL_ALWAYS, GL_LESS, GL_GREATER osv ...

ref: referanseverdi som alpha verdien testes mot.

Stencil Buffer

- Brukes hovedsaklig for å begrense rendering til utvalgte deler av framebufferet.
- For å ta i bruk stencil bufferet må man aktivere stencil test:

glEnable(GL_STENCIL_TEST);

glDisable(GL_STENCIL_TEST);

- Man kan styre oppdatering av stencil bufferet på bitnivå
  - glStencilMask(Gluint mask);
- Testen styres ved

glStencilFunc(Glenum func, Glint ref, Gluint mask);

func: GL_NEVER, GL_ALWAYS, GL_LESS, GL_GREATER osv ...

ref: referanseverdi som stencil verdien testes mot.

Accumulation Buffer

- Accumulation-bufferet brukes for å akkumulere fargeverdier basert på flere iterasjoner av rendering til et fargebuffer.
- Når akkumuleringen er ferdig så leses verdiene tilbake til et fargebuffer for visualisering.
- Antialiasing, motion blur, depth of field
- Demo: Particles_accumulation på itsl

Accumulation Buffer (cont)

- For å bruke accumulation buffer brukes glAccum(Glenum op, Glfloat value)

op: GL_ACCUM, GL_LOAD, GL_RETURN, GL_ADD, GL_MULT

Se rødboka for detaljer

- GL_ACCUM: Leser verdier fra backbuffer, multipliserer med value og legger svaret til i accumulation bufferet
- GL_LOAD: Leser verdier fra backbuffer, multipliserer med value og copierer svaret til accumulation bufferet
- GL_RETURN: Leser verdier fra accumulation bufferet, multipliserer med value og copierer svaret til backbufferet
- GL_ADD/GL_MULT: manipulerer verdiene i accumulation bufferet

Polygon Offset

- Om man skal tegne geometri samtidig som man ønsker å tegne wireframe modellen av geometrien, så kan man få problemer med dybdetesten. Dybdetesten vil ikke vite hva slags piksler som ligger fremst.

- For å løse dette kan man bruke *polygon offset* for å legge til en *offset* som sørger for at sammenfallende z-verdier blir adskilt.

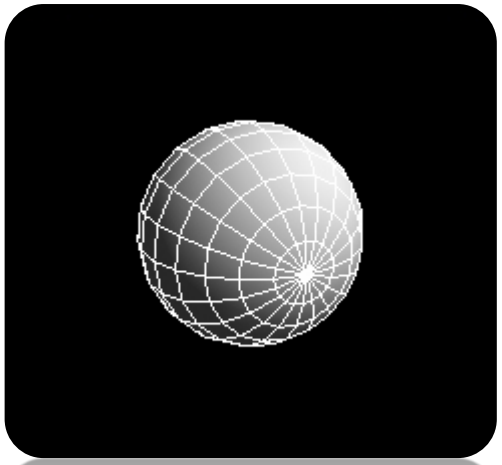
- glEnable(GLenum pname);

  GL_POLYGON_OFFSET_FILL,

  GL_POLYGON_OFFSET_LINE,

  GL_POLYGON_OFFSET_POINT

- glPolygonOffset(...);



Lab

- Hjelp med Mappe 2