# PG430 – Innføring i grafikkprogrammering

Forelesning 4

André R. Brodtkorb
Andre.Brodtkorb@nith.no

# Oversikt

- Repetisjon fra forrige time
- Farger
- Lysegenskaper
- Materialegenskaper
- Lab: Mappeinnlevering

# Mappe 1 hint

- Når du skriver mappene i kurset:
  Keep it simple!
  - http://en.wikipedia.org/wiki/KISS_principle

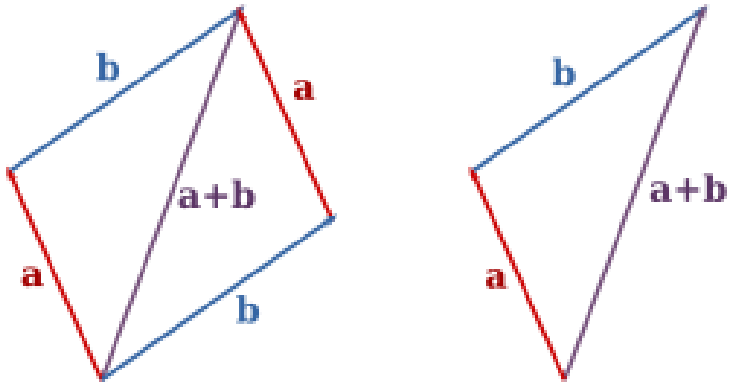
    *The KISS principle states that simplicity should be a key goal in design, and that unnecessary complexity should be avoided.*

# REPETISJON

# Vektor operasjoner

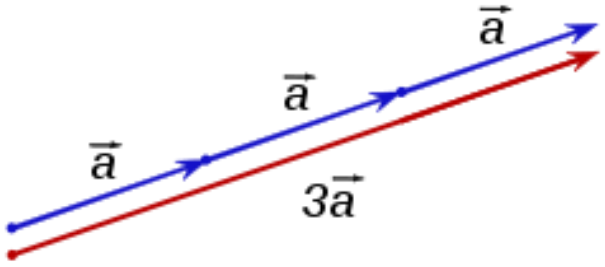
- Addisjon:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_x + b_x \\ a_y + b_y \\ a_z + b_z \end{pmatrix}$$




- Skalering:

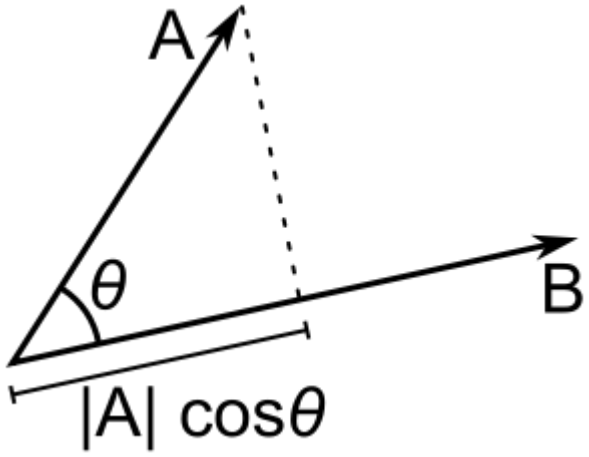
$$s\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} sa_x \\ sa_y \\ sa_z \end{pmatrix}$$



# Vektor operasjoner

- Prikk-produkt (dot product):

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \cdot \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x * b_x + a_y * b_y + a_z * b_z$$

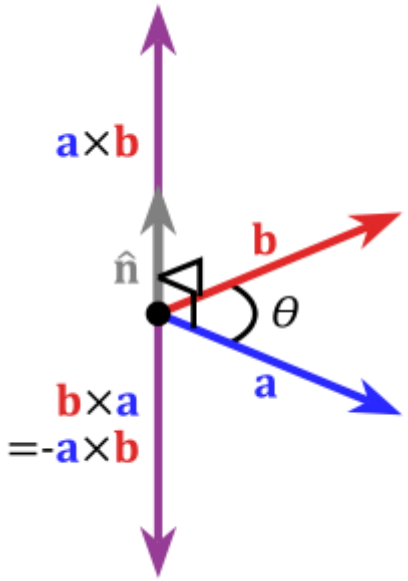



  - Kan tolkes som "lengden av A i retning B", eller "A projisert ned på B"
  - Brukes bl.a. til lyssetting.
  - Er null dersom a står vinkelrett på b

# Vektor operasjoner

- Kryss produkt:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - a_z b_y \\ -(a_x b_z - a_z b_x) \\ a_x b_y - a_y b_x \end{pmatrix}$$




  - $a \times b$ står vinkelrett på både a og b.
  - Er null dersom a og b er parallelle
  - brukes i utregning av normaler

# Homogene koordinater

- For å også tillate translasjoner bruker OpenGL 4x4 matriser:

$$\begin{pmatrix} a & d & g & t_x \\ b & e & h & t_y \\ c & f & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} ax + dy + gz + wt_x \\ bx + ey + hz + wt_y \\ cx + fy + iz + wt_z \\ w \end{pmatrix}$$

- Slike transformasjoner kalles affine
  - Lineær transformasjon plus translasjon
- For punkter er w=1
- For vektorer er w=0.
  - Dette er fordi vektorer ikke skal translateres, men er retninger

# Modeling-transformasjoner

- Object space:
  - Vertex posisjoner og normaler er definert i et koordinatsystem som kalles *object space*.
  - Alle objekter kan defineres i et eget unikt object space.

- World space:
  - Vi trenger et felles koordinatsystem som kan inneholde en mengde forskjellige objekter. Dette koordinatsystemet kalles *world space*.

- Modeling transformation:
  - Transformasjonen mellom object space og world space kalles *modeling transformation*.

# Viewing-transformasjoner

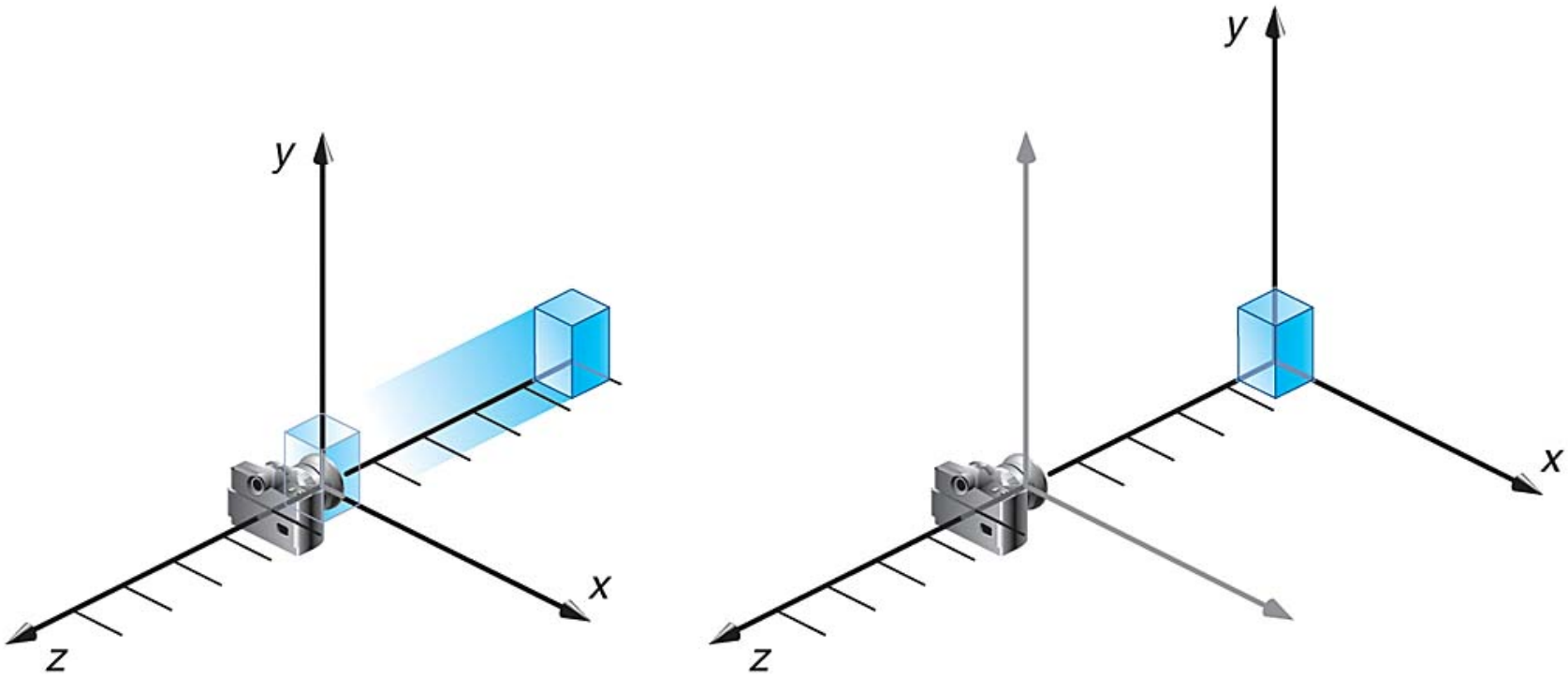
- Eye space:
  - Kameraet er definert i et koordinatsystem som kalles *eye space*.

- Viewing-transformation:
  - Transformasjonen mellom *world space* og *eye space* kalles *viewing transformation*.

- I OpenGL er *model matrix* og *viewing matrix* kombinert i en og samme matrise som kalles *modelview matrix*.

# Kamera

- Det er to måter å se på modelview-transformasjoner i OpenGL.
  - Flyttes kamera?
  - Flyttes objektene?



# Rekkefølgen på transformasjoner

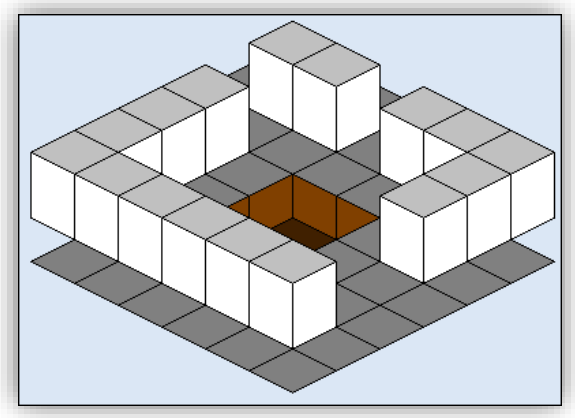
| | |
|---|---|
| `glMatrixMode(GL_MODELVIEW);` | switch to `MODELVIEW` |
| `glLoadIdentity();` | `MODELVIEW` matrix $M$ is $I$ |
| `glMultMatrix*(`$M_n$`)` | $M = M_n$ |
| `glMultMatrix*(`$M_{n-1}$`)` | $M = M_n M_{n-1}$ |
| $\vdots$ | $\vdots$ |
| `glMultMatrix*(`$M_2$`)` | $M = M_n M_{n-1} \cdots M_2$ |
| `glMultMatrix*(`$M_1$`)` | $M = M_n M_{n-1} \cdots M_2 M_1$ |
| | |
| `glVertex*(p)` | $\mathbf{p}' = M_n M_{n-1} \cdots M_2 M_1 \mathbf{p}_1$ |

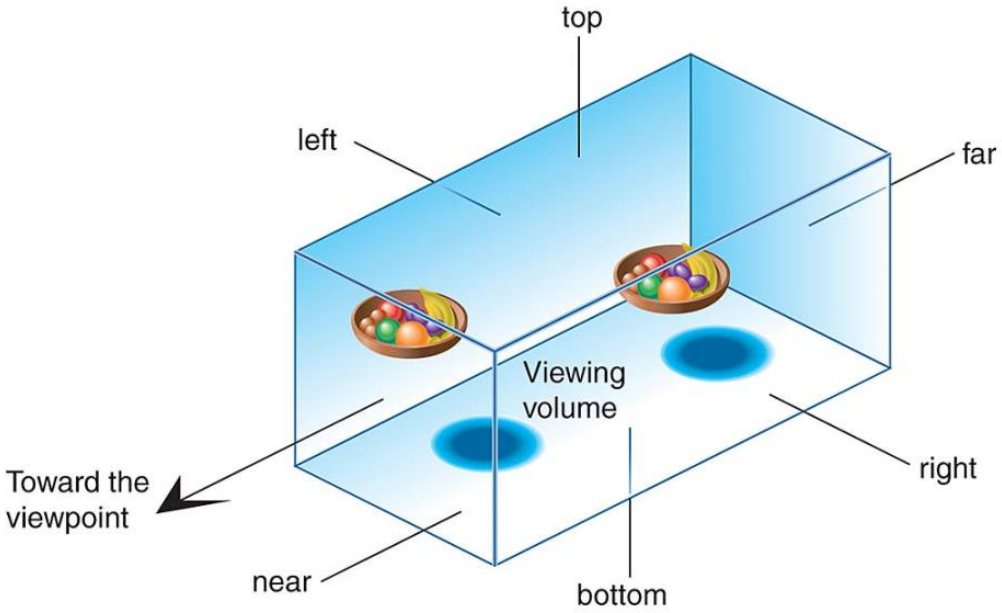
- Første transformasjon som utføres på vertexen er siste matrise angitt i koden
- Gjelder også glTranslate*, glRotate* ...

# Ortografisk projeksjon

- Ved ortografisk projeksjon har *viewing volume* form som en boks.

glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far);

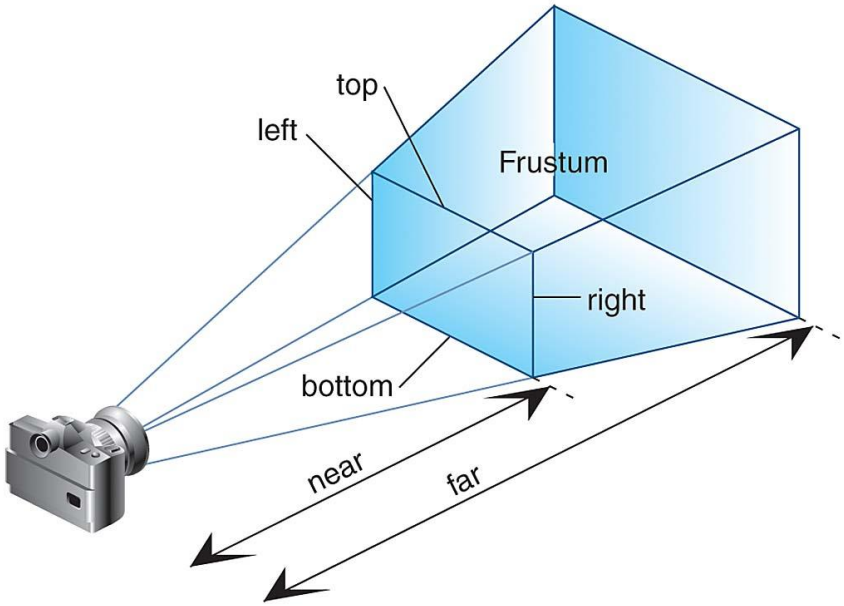




# Perspektiv projeksjon

- Objekter som er nært kamera blir større enn objekter som er lengre unna kamera.

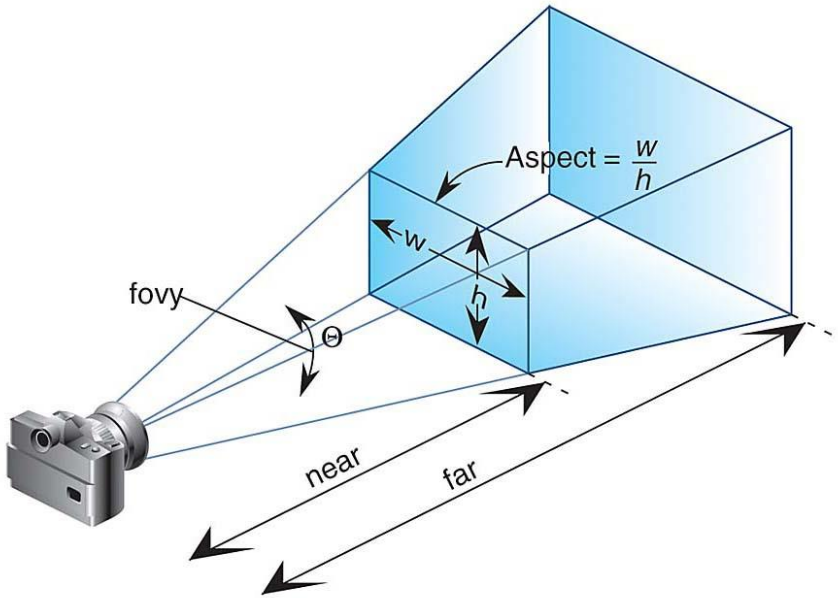
glFrustum(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far);

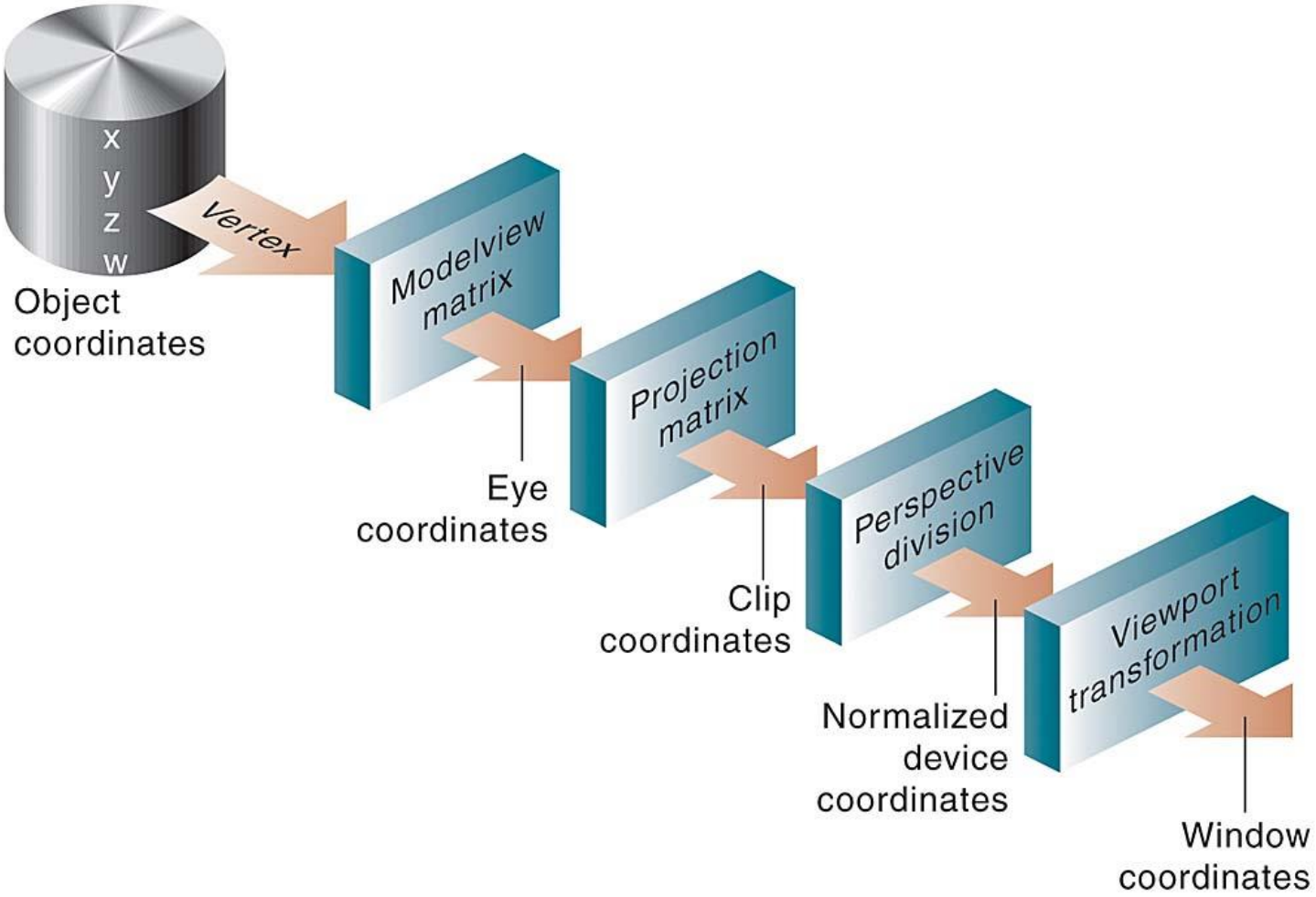
# Perspektiv projeksjon

- Om man synes glFrustum er vanskelig å bruke kan man bruke følgende kommando fra *glu* i steden for:

```
gluPerspective(GLdouble fovy, GLdouble aspect,
      GLdouble near, GLdouble far);
```



# Transformasjoner forts.

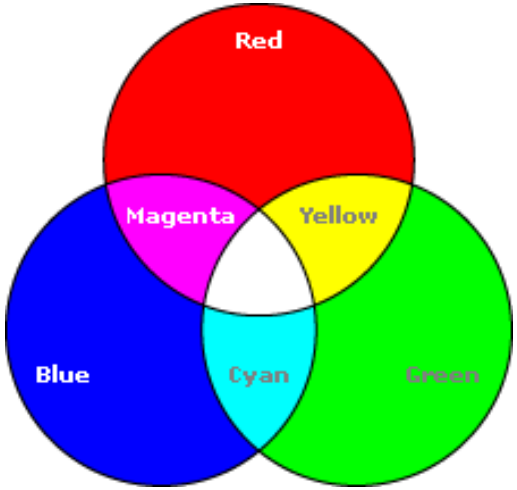



Koordinatsystemer og transformasjoner i OpenGL

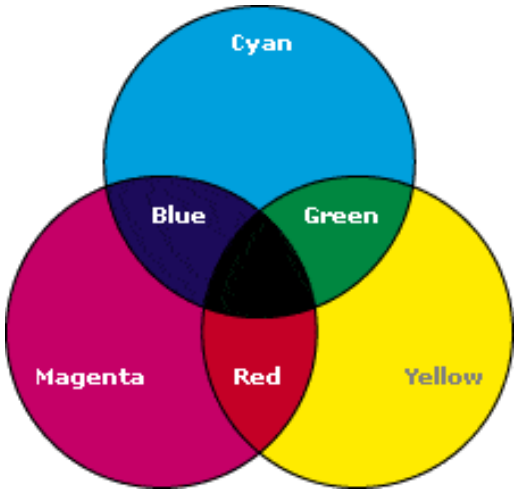
# DAGENS FORELESNING

# Farger i OpenGL

- I OpenGL spesifiseres en farge som separate intensiteter av rød, grønn og blå komponenter.
- RGB er basert på en adderende fargeblanding.
- Det finnes andre fargemodeller som CMYK (cyan, magenta, yellow, black) som er basert på en subtraherende fargeblanding (blekkskrivere)

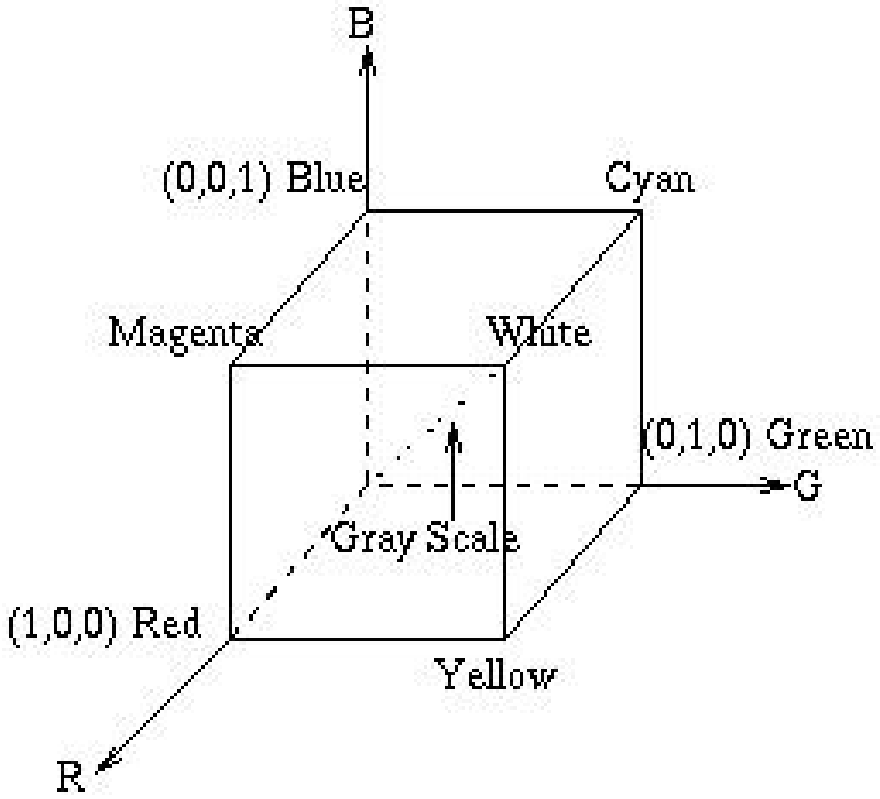


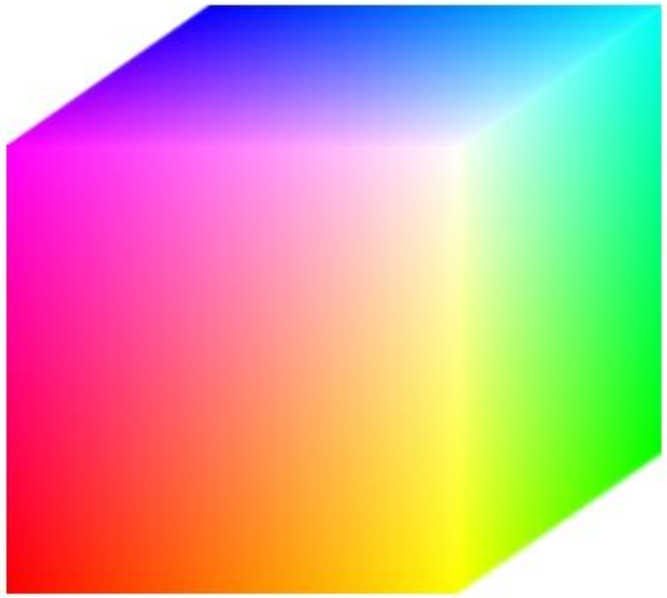

Additiv - OpenGL




Subtraktiv - Printing

# Fargekuben til OpenGL



# Funksjon for å spesifisere farge

- Denne funksjonen setter fargen til alle vertexer som tegnes etterpå.

glColor{3,4}{b,d,f,i,s,ub,ui,us}(red, green, blue, alpha);

# Shading Model

- Smooth shading (Gouraud shading)
  - Fargeverdiene blir interpolert mellom vertexene i et primitiv
  - glShadeModel(GL_SMOOTH);

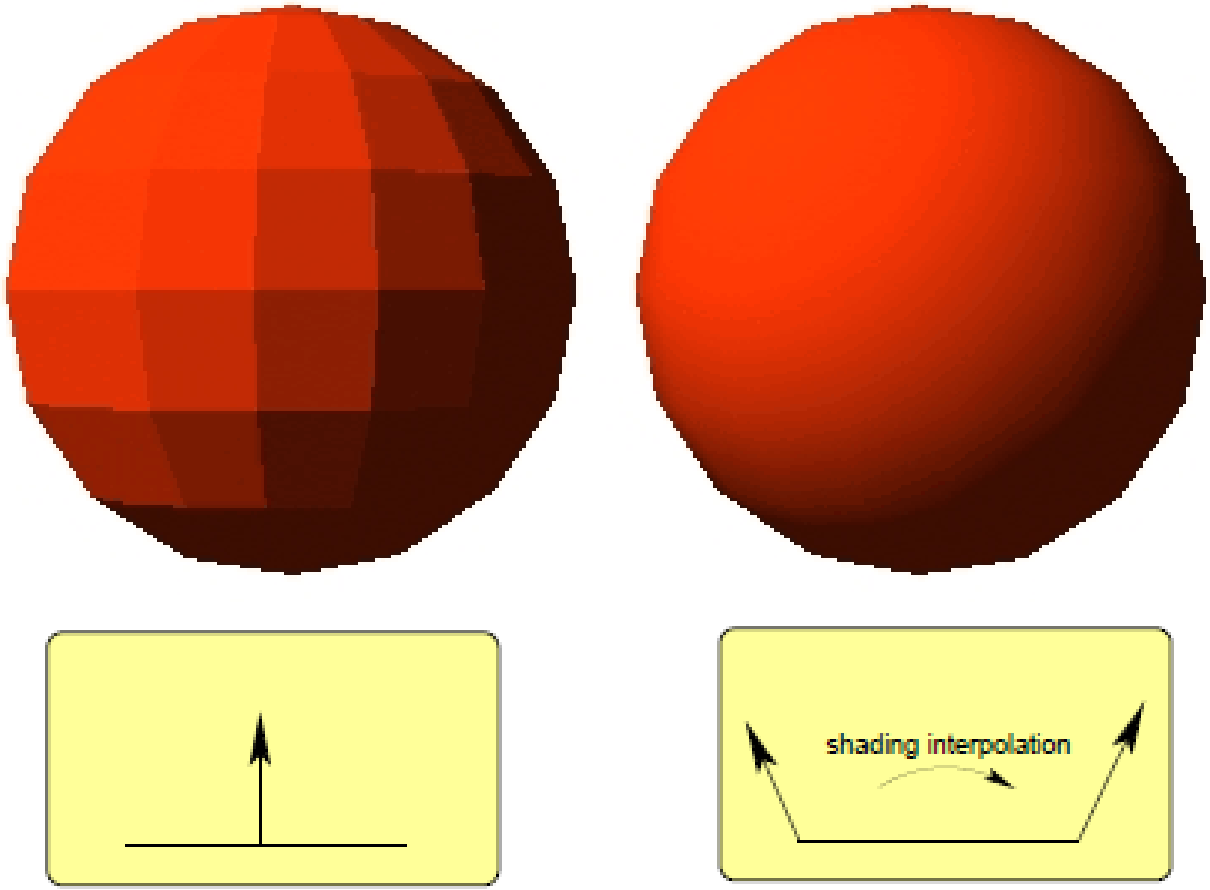
- Flat shading
  - Ingen shading blir utført i et primitiv. Fargen til primitivet bestemmes som regel av fargen som var satt for den siste vertexen i primitivet.
  - glShadeModel(GL_FLAT);

# Demo - Video

- Henri Gouraud, 1971: Gouraud shading

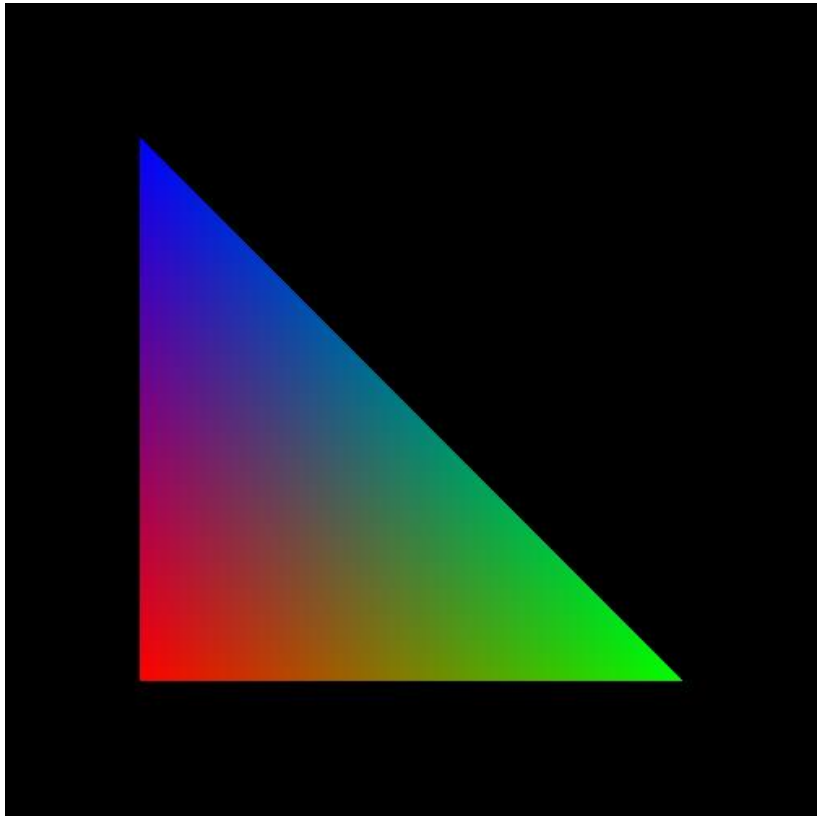
- http://nerdplusart.com/first-3d-rendered-film-from-1972-and-my-visit-to-pixar

# Flat shading vs. smooth shading



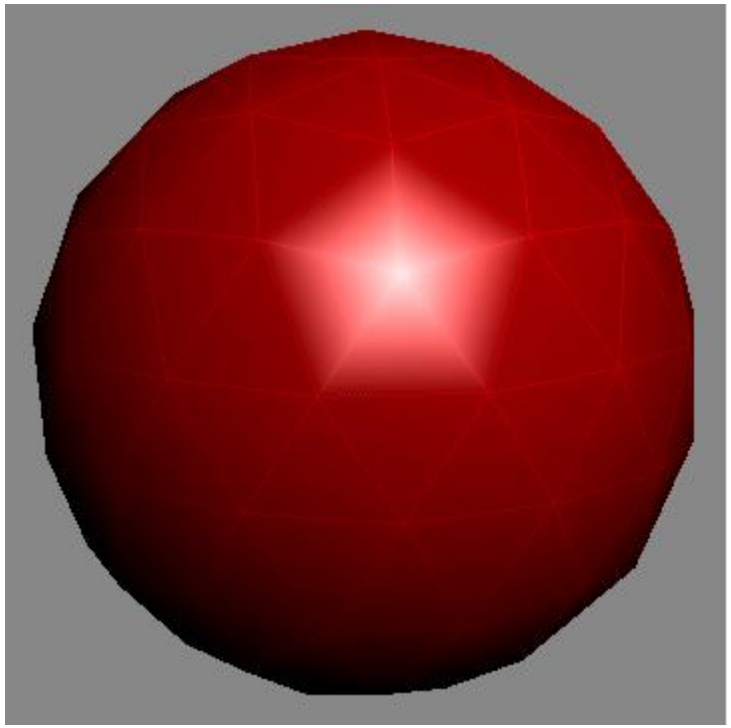
# Smooth shading av et triangel

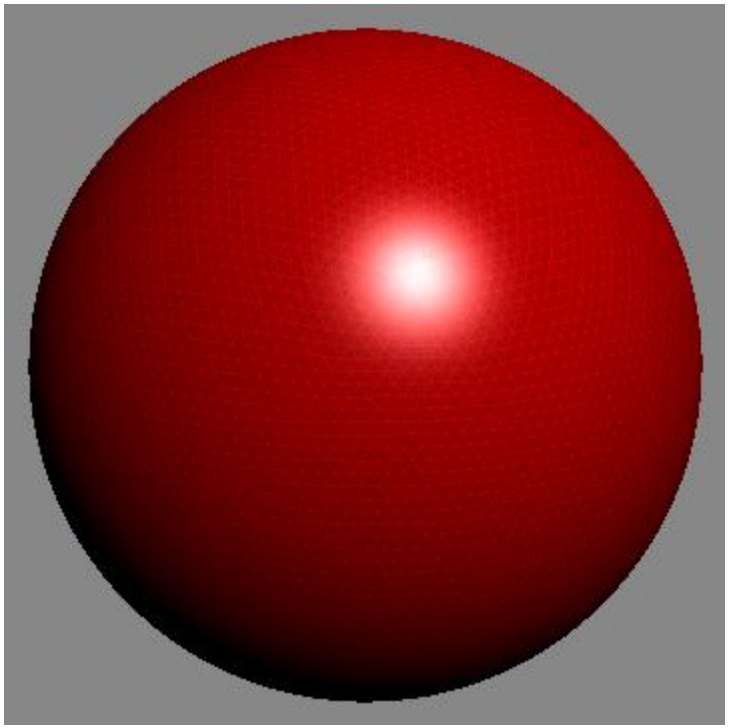
```
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glShadeModel(GL_SMOOTH);

glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLES);
  glColor3f(1.0f, 0.0f, 0.0f);
  glVertex2f(-1.0f, -1.0f);
  glColor3f(0.0f, 1.0f, 0.0f);
  glVertex2f(1.0f, -1.0f);
  glColor3f(0.0f, 0.0f, 1.0f);
  glVertex2f(-1.0f, 1.0f);
glEnd();
```



# Smooth shading

- Gouraud shading har fortsatt artifakter ved lav poly-count





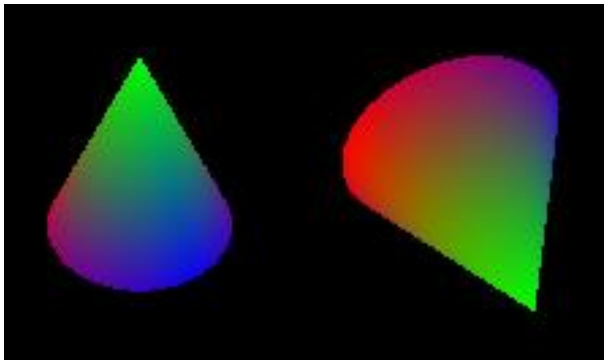
Images from Wikipedia, user Jalo

# Hidden Surface Removal

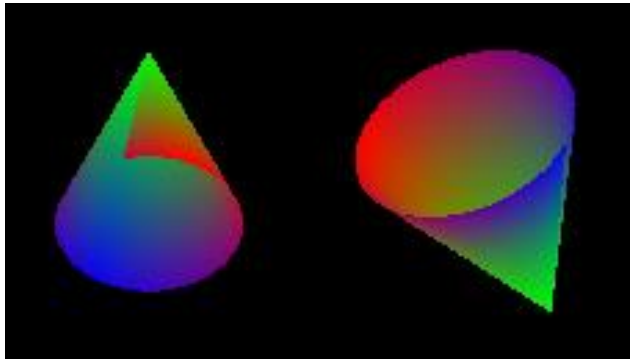
- Når man begynner å tegne objekter som er tredimensjonale er det viktig å tegne objekter som befinner seg foran andre objekter samtidig som man fjerner objekter som er skjult.

- Hvis man ikke har en mekanisme for å håndtere dette kan man få uønskede artefakter.

- Løsningen på dette er å bruke noe som heter dybdetest.



Med dybdetest



Uten dybdetest

# Hidden Surface Removal forts.

- Et dybdebuffer tar vare på dybden til en pixel, det vil si avstand i forhold til kamera.

- Hvis en pixel har en mindre dybdeverdi enn en tidligere tegnet pixel blir den tegnet, hvis dybdeverdien er større blir pixelen forkastet.

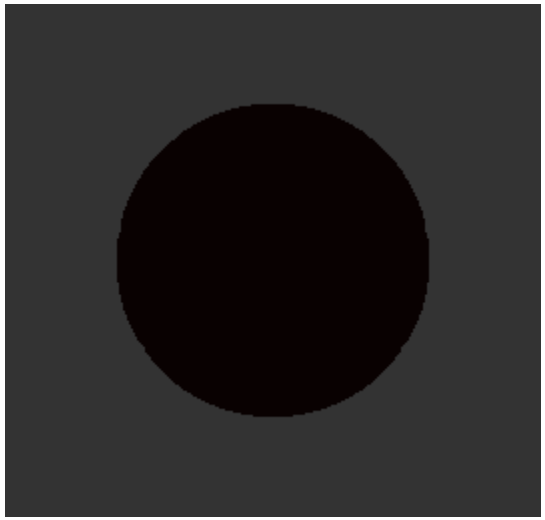
```
void init() {
    SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
    glEnable(GL_DEPTH_TEST);
}
void render() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    …
}
```

# Lyssetting

# Lyssetting

- For å skape en 3-dimensjonal scene er det viktig med lyssetting.
- I dette tilfellet ser man ikke forskjell på en ball og en tallerken når lyssetting ikke brukes.

Lyssatt



Ikke lyssatt

# Phong Reflection Model

- OpenGL bruker såkalt Phong Reflection Model
  - Oppkalt etter doktoravhandlingen til Bui Tuong Phong 1973.

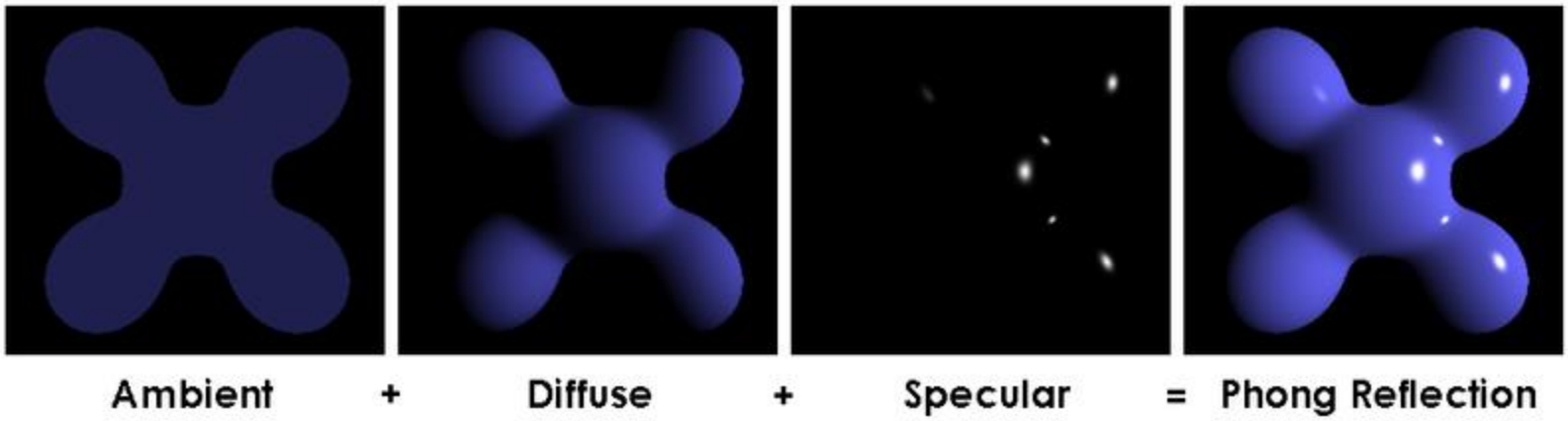
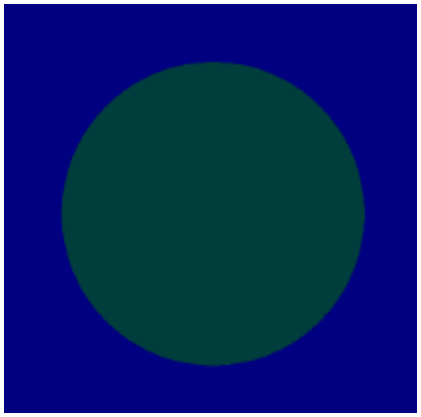



Image from Wikipedia, user Rainwarrior

# Lysegenskaper – Ambient Light

- Ambient lys har ingen retning.
- Objekter som blir opplyst med ambient lys blir lyssatt likt på alle overflater og i alle retninger.
- En approksimasjon av "sekundærlys"

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient)

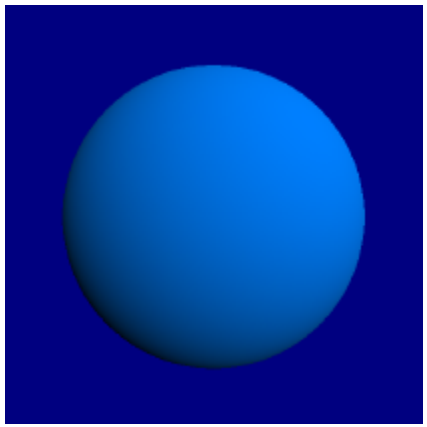


Ambient lys gir samme farge over hele primitivet.

# Lysegenskaper – Diffuse Light

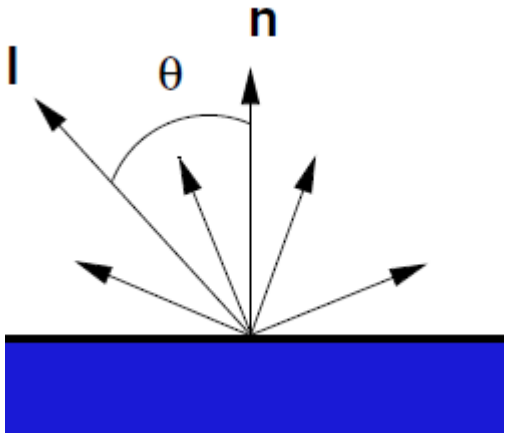
- Diffust lys kommer fra en lyskilde, og er avhengig av normalene til objektet.
- Uavhengig av synsretning
- En approksimasjon av lyssetting av en matt overflate

glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse)

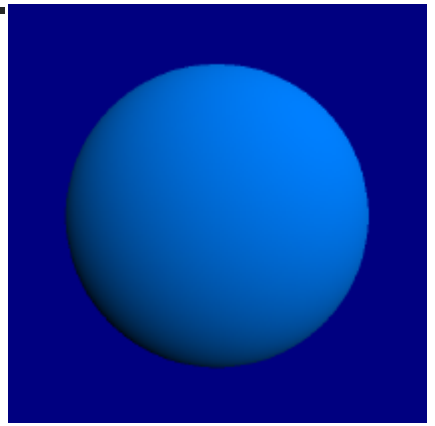


Her står lyskilden til høyre for objektet og det diffuse lyset bidrar dermed til lyssettingen av den høyre halvkulen.

# Diffuse light




- Regnes ut ved å bruke prikkproduktet mellom normal-vektoren og lysvektoren

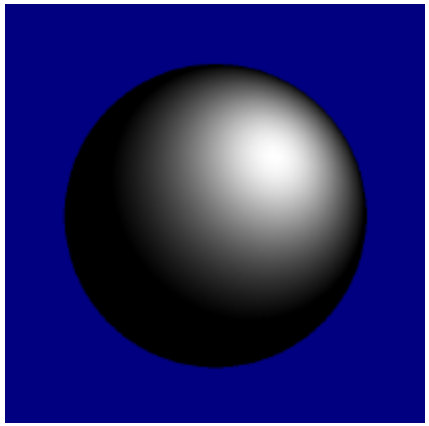


- diff = max(n*l, 0)

# Lysegenskaper – Specular Light

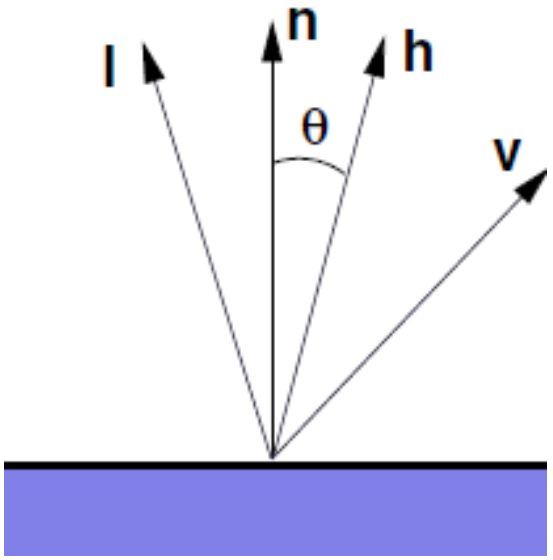
- Specular lys kommer fra en spesiell retning og er avhengig av normalen til objektet samt kameraposisjon.
- En approksimasjon av lyssetting til en blank overflate

```
glLightfv(GL_LIGHT0, GL_SPECULAR, light_spec)
```

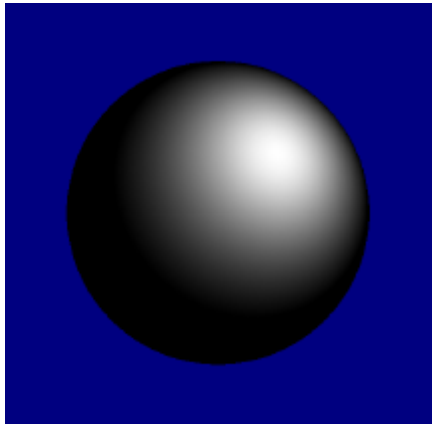


Specular lyssetting av kula.

# Specular light

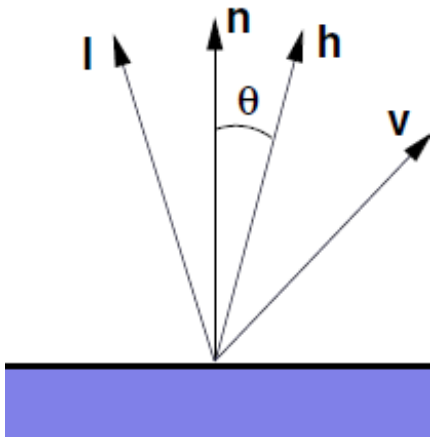


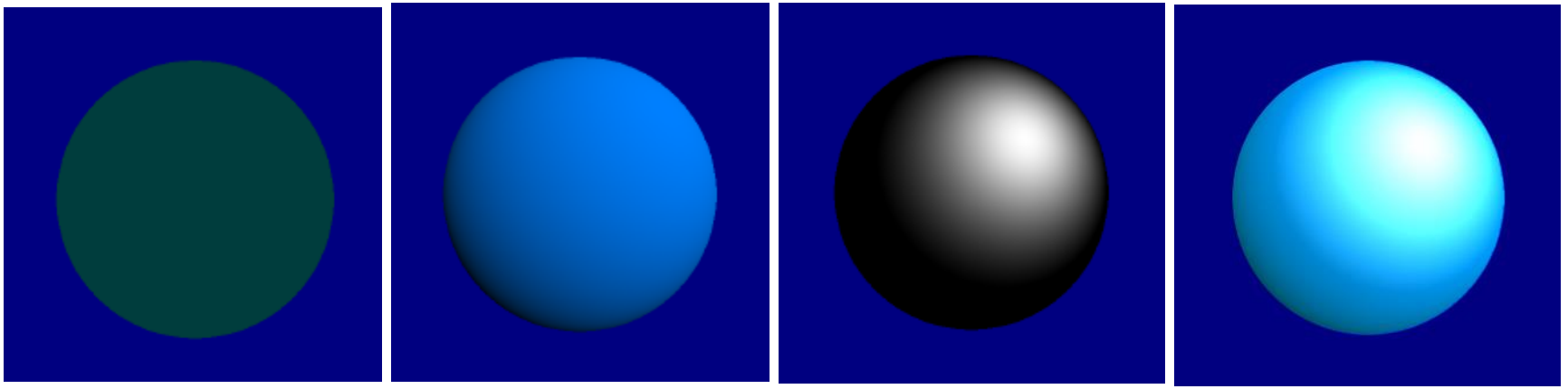

- Regnes ut ved prikkprodukt mellom halv- og view-vektor
- I tillegg brukes en "shininess" parameter for å si hvor skarp refleksjonen skal være
- spec = max(h*n, 0)^shininess



# Lyssetting oppsummert

- light = ambient
  + max(n*l, 0)
  + max(h*n, 0)^shininess



Ambient + Diffuse + Specular = Phong Lighting

# Lyskilder og materialer

- OpenGL tillater flere samtidige lyskilder
  - glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    …
- Lysets egenskaper sammen med matrialegenskapene bestemmer den endelige fargen fargen
  - Eks: Rød ball i hvitt lys blir rød
  - Rød ball i grønt lys blir sort

# Materialegenskaper

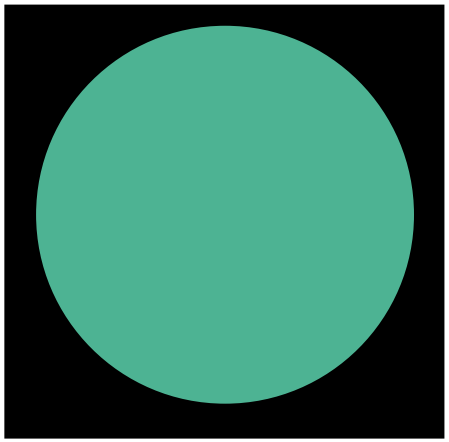
- På samme måte som lys, har materialer forskjellig ambient, diffuse og specular farge som brukes for å bestemme materialets refleksjoner.

- glMaterialfv(GL_FRONT, GL_EMISSION, mat_emis);
- glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
- glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
- glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

# Materialegenskaper - Emission

- Emission er en materialegenskap for å angi hvor ”selvlysende” et objekt er.
- Blir ikke regnet som en lyskilde (gjelder selv om lyssetting er disablet)

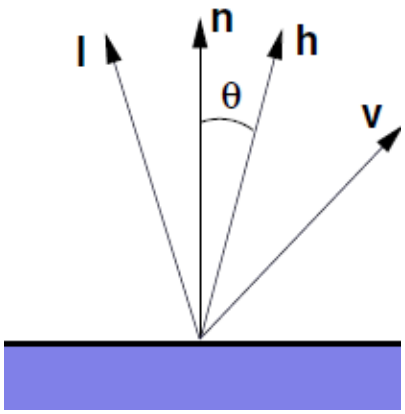
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emis)

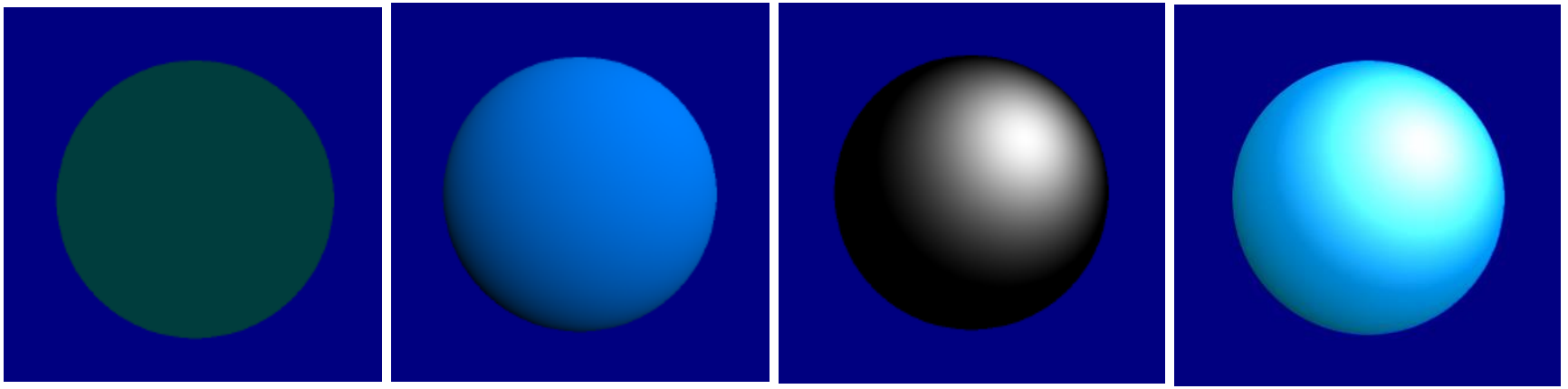


Kula får en cyan farge selv om den ikke blir lyssatt. Denne effekten kan brukes for f.eks. lamper.

# Lyssetting med materialegenskaper

- color = M_emission
  + L_ambient * M_ambient
  + max(n*l, 0) * M_diffuse
  + max(h*n, 0)^shininess * M_specular



Ambient + Diffuse + Specular = Phong Lighting

# Color Material

- glMaterial er en dyr operasjon
- For å øke ytelsen kan man bruke *color material*. Da endrer glColor en materialegenskap og ikke fargen.
- Må aktivere *color material* med kommandoene:

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GLenum face, GLenum mode);
```

*face* angir hvilken side av et primitiv man skal påvirke med kommandoen mens *mode* angir hvilke typer materialegenskaper som skal oppdateres.

# Color Material forts.

- Kodeeksempel:

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
// glColor forandrer nå diffuse refleksjon
glColor3f(0.2f, 0.5f, 0.8f);
// tegn geometri
glColorMaterial(GL_FRONT, GL_SPECULAR);
// glColor forandrer nå specular refleksjon
glColor3f(0.9f, 0.0f, 0.2f);
// tegn geometri
glDisable(GL_COLOR_MATERIAL);
```

# Demo – Nate Robins

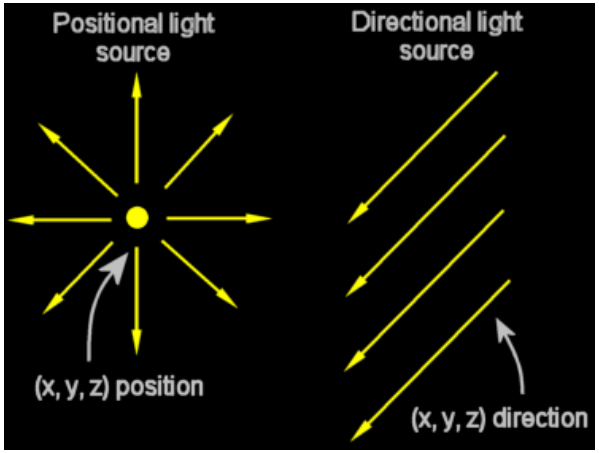
- Last ned nate_robins.zip fra ”it’s learning”
- Kjør lightmaterial.exe
- Prøv å sett forskjellige lys- og materialegenskaper og observer hvordan disse påvirker lyssettingen av objektet.

# Lysposisjon

- Ved å sette posisjonen til en lyskilde spesifiserer man også hvilken type lys man har.

```
glLightfv(GL_LIGHT0, GL_POSITION, light_pos)
```

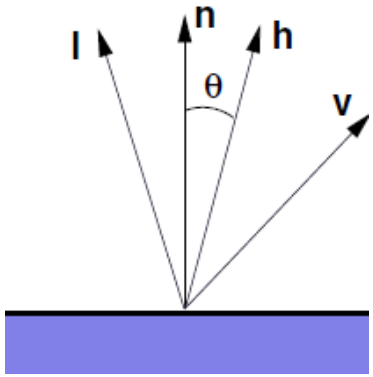
- Directional light (sol)
  - GLfloat light_pos[] = {1.0, 1.0, 1.0, 0.0}
  - Alle objekter lysettes likt!

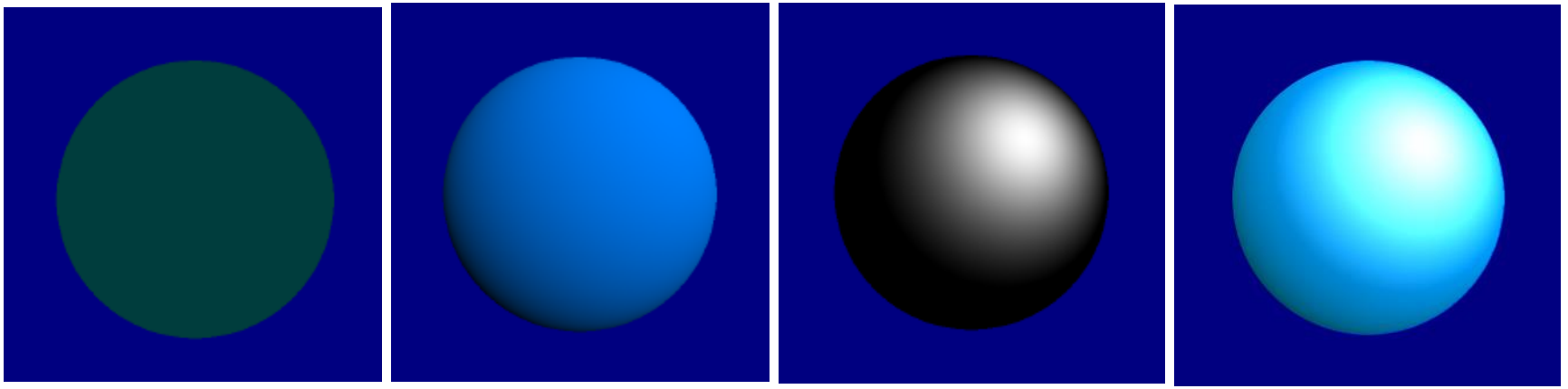



- Positional light (lampe)
  - GLfloat light_pos[] = {1.0, 1.0, 1.0, 1.0}
  - Alle objekter lyssettes forskjellig avhengig av posisjon

# Lyssetting med avstand

- color = M_emission
  + C_amb
  + (C_diff + C_spec) * atten
- atten = 1 / (a + bd + cd^2) (quadratic)
  Light gets weaker as the distance get larger



Ambient + Diffuse + Specular = Phong Lighting

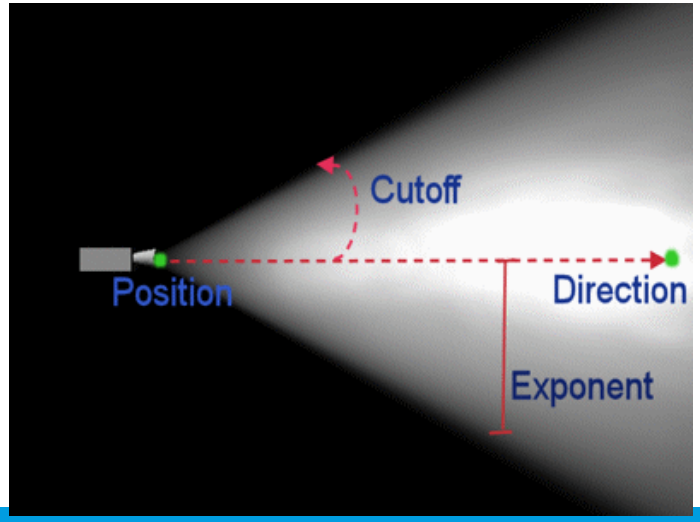
# Spotlights

- Spot direction angir hvor spotlight'en peker

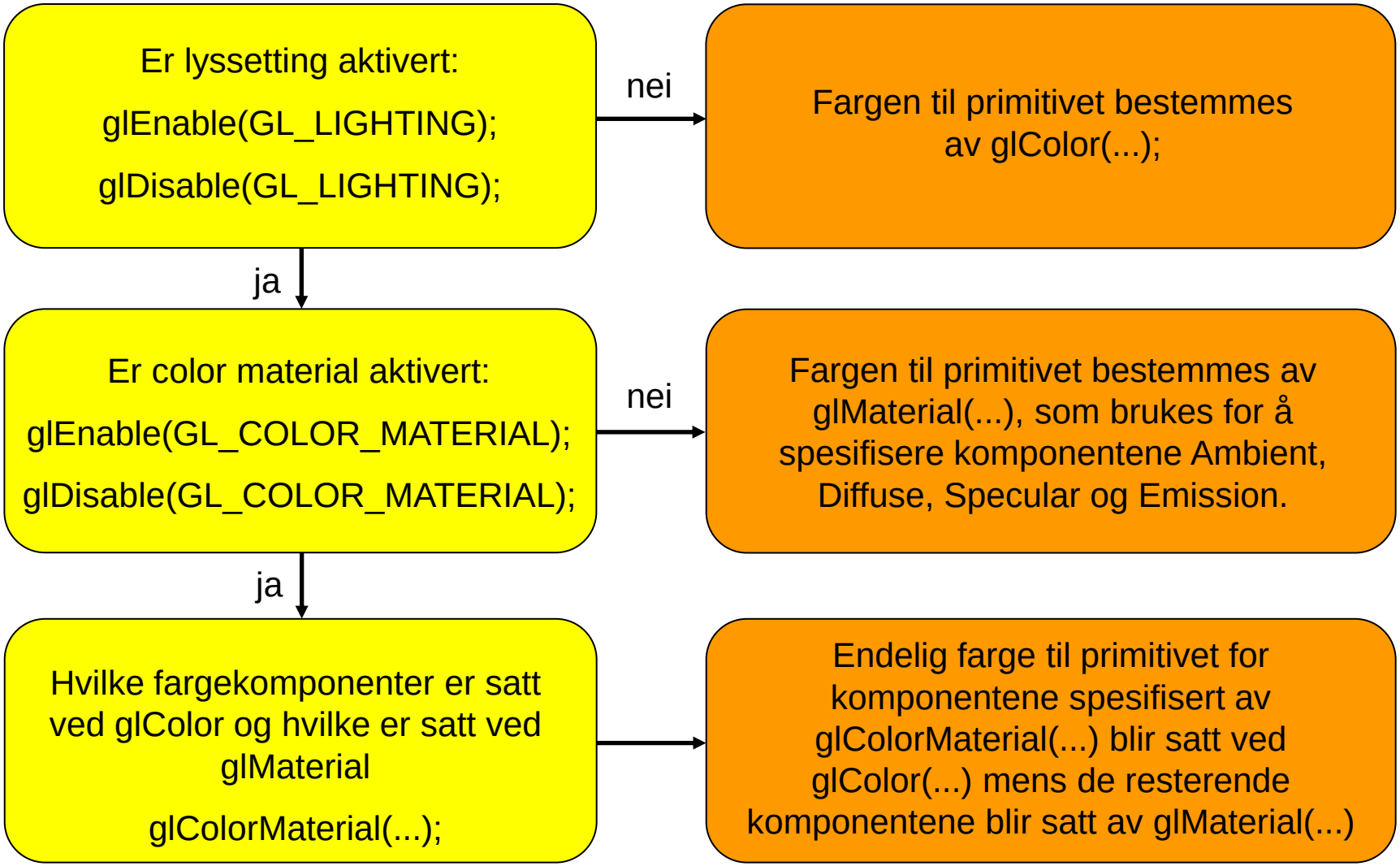
  glLightf(GL_LIGHT0, GL_SPOT_DIRECTION, dir); dir: [x, y, z]
- Spot cutoff angir bredden på en kjegle som definerer en spotlight.

  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, angle); angle: [0-90]
- Spot exponent brukes for å angi hvor konsentrert lyset er.

  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, exp); exp: [0-128]



# State-flytdiagram



# Demo – Nate Robins

- Kjør lightposition.exe

- Prøv å forandre på lysposisjon og den siste komponenten som setter om lyset er directional eller positional.

# Attributt stack

- I tillegg til matrise stack har OpenGL en attributt stack
- Nesten alle tilstander (states) kan lagres
- Eksempler:
  - Lagrere lys egenskaper og matrialegenskaper:
    - glPushAttrib(GL_LIGHTING_BIT)
  - Lagre dybde test tilstand:
    - glPushAttrib(GL_DEPTH_BUFFER_BIT)
  - Lagre begge:
    - glPushAttrib(GL_DEPTH_BUFFER_BIT | GL_LIGHTING_BIT)
  - Sett tilbake den lagrede tilstanden
    - glPopAttrib();
- Merk: Alle attributter lagres i samme stack
- Kan være veldig inneffektivt

# PAUSE – 15 MIN

# Lab

- Help med Mappe 1
- Hjelp med tidligere lab-oppgaver

# Lab: Lyssetting

- Ta utgangspunkt i kode og lag fire forskjellige lys (rød, grønn, blå og hvit) som man kan veksle mellom med tastene r, g, b og w.
- Prøv å veksle mellom directional og positional lysegenskaper.
- Forandre materialegenskapene til objektet for å få objektet til å se ut som f. eks. gull eller svart gummi (finn parametere med lightmaterial.exe).
- Eksperimenter med forskjellige parametere for positional light.

# Lab: Bruk to lys med forskjellig farge samtidig.