DAT155 Graphic methods

last updated 28.09

All exercises are curriculum

Exercise 2

Requirement:

Submission deadline Friday 18/9.

The programs for assignments 2 and 5 have to be demonstrated for the lab assistant. The source code and video (ordinary format) have to be submitted for these two assignments. Use only one zip-file.

No submission for assignments 1 and 3. Every group print out one model. Demonstration on Tuesday 8/9 or 15/9 at the lab.

Purpose

Scanning - > point cloud -> Mesh - > 3d printing
Hierarchical model
Transformations
Camera system

Assignment 1 (Demonstration)

Scanning - > point cloud -> Mesh - > 3d printing

Veiledning vil bli gitt:

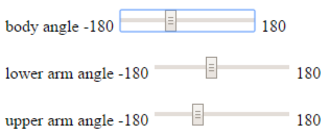
I denne oppgave demonstreres scanningt vha. en håndscanner. Av bla. hensyn til smittevern, så gjør vi det slik i år.
Programvaren for håndskanneren lager en punktsky (point cloud, a set of data points i space). Punktskyen tar vi inn i et program, blender eller meshmixer som lager et mesh fra en punktsky. Et mesh har flater. Mesh'et kan vi endre på ved f.eks. å redusere antall vertices (hjørner).
Deetter lagrer vi mesh'et i et vanlig format (f.eks. obj.). Videre må vi så ta mesh'et (modellen) inn programmet for 3D-printeren som er makerbot for å forberde for 3D-printing. Videre lagrer vi mesh'et i et bestemt format som 3D-printeren (Makerbot Replicator Z-18) kan tolke (med nødvendig info for printing). Dersom det er kapasitetsproblemer med scanningen, kan hver gruppe ta ned en liten modell av en figur i vanlig format (f.eks. obj) som dere 3-printer.

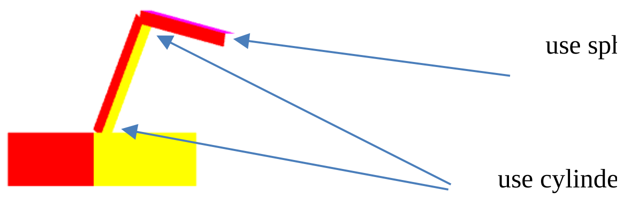
Dere kan gjøre litt forberedelser før lab-dagen.
På lab-dagene 1.9 og 8.9 blir det en demo-øving i 3D-scanning og 3D-printing
Dette kommer i tillegg til programmeringsoppgavene:
Pga bla. smittevern osv. må vi ta scanningen med håndscanneren som demo.
Dere kan forberede dere litt å denne oppgaven.
Dere kan gjøre litt forberedelser til lab-dag på forhånd.

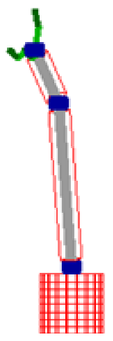
Meshmixer eller blender er et gratisprogram
1. meshmixer er et gratisprogram man kan bruke til å hente inn en punktsky fra scanningen og få ut et mesh (rutenett av veretex'er)
http://www.meshmixer.com/ (Lenker til en ekstern side.) (Lenker til en ekstern side.)
Behøver ikke installere denne for denne oppgaven siden vi tar scanningen som demo.
Alt: Blender
https://www.blender.org/ (Lenker til en ekstern side.)
2. Programvare for 3d-printeren:
https://www.makerbot.com/de/3d-printers/apps/makerbot-print/download/ (Lenker til en ekstern side.) (Lenker til en ekstern side.) En i gruppen installerer denne på forhånd.
3. En i hver gruppe tar med en minnepenn
4. Hver gruppe henter ned en lovlig mesh (modell) for å printe ut gjerne i obj-format.
 Modellen må tas inn i makerboot-programmet:
 OBS! Sjekk at modellen står "støtt" på planet i makerboot-programmet.
 Skaler modellen til 5-10 cm lelers tar det for lang tid å skrive ut.
5. Finne ut hva et mesh er for noe:https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/What_is_a_Mesh%3F (Lenker til en ekstern side.) (Lenker til en ekstern side.)
6. Se gjennom slides fra den andre boken: chapter 3 (modeling)
7. Se denne:

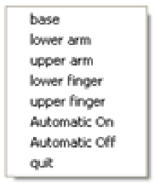
https://www.3dhubs.com/guides/3d-printing/

Assignment 2 (Demonstration and submission)









Run the project **robotArm** in **chap9**. The first figure display the robot arm. Study the source code, the text and the figures 9.8 and 9.9 in the textbook.

We want to **expand** the robot arm with **a claw** and possible joints. See the second figure.
The claw may have two fingers. A finger can have two parts.
We may want to use joints for the robot arm, a cylinder between the base and the lower arm, a cylinder between lower arm and upper arm, a sphere between the upper arm and the claw.
Remember, it will be optionally to have joints. It is recommended first to build the robot without these joints.

All sub objects are solid and shall not intersect each other.
The threshold values for rotation angles must therefore be appropriate (not $\pm 180^0$).

The rotation is about the cylinder main axis when a cylinder joint is used.
The rotation is about x, y or z- axes when a sphere joint is used.

You can use a cylinder for the base if you want. The base should be able to rotate about its z-axis and the rest of the robot arm has to follow naturally.

Furthermore, we want to highlight the part that is under manipulation visually. To do this one can use a uniform variable for each part object. Use a vec3 to put a color value (RGB) as a uniform variable.

We also want to move the entire model by translation in x-z plane using keys.
If you want, the animations can be controlled by keys.
A source code for cylinder with top and bottom will be placed in the folder for the exercise:
You will find a source code for a sphere, **wireSphere** (use fill) is in **06**.
Remarks:
To save and restore model-view matrix if necessary we can use push or pop this matrix.
Study the paragraph 9.4.1 in the textbook.
push and pop are part of Array object in js (Ref: Angel)

```
var stack = [ ]
stack.push(modelViewMatrix);
modelViewMatrix = stack.pop();
```

The left finger consists of finger1 and finger2, same for the right finger.
Before we draw left_finger1, you can push the model matrix at the join. When we finished with left_finger2 we pop the model matrix and we will "return back" to the join and you is ready to do the same operations for the right-finger.
You will find geometry for other objects here:
http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SEVENTH_EDITION/CODE/GEOMETRY/
http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SEVENTH_EDITION/CODE/GEOMETRY/geometryTest.html

Alle må lage en klo med to (eller flere fingre) plassert oppå overarmen. Hver finger skal ha to deler. Fingrene kan beveges mot hverandre. Bygg gjern videre på den koden dere laget i Exercise 1 (javascript 6 og webgl 2.0), men det også greit om dere fortsetter på koden robotarm i 09.
De ulike delene skal ikke skjære hverandre (gå inn i hverandre).
Det er frivillig å lage fysiske ledd. Ideelt sett bør det være sylinderledd mellom base og underarm og mellom underarm og overarm (2 frihetsgrader av rotasjon) og kuleledd mellom overarm og klo (3 frihetsgrader for rotasjon).

Frivillig: Dere kan vurdere å bruke taster på bevegelsene. Det er da ønskelig å redusere antall taster man beveger roboten til et mindre antall. For å oppnå det kan vi velge en robotdel om gangen.
Vurder å benytt en enumerert liste av robotmanipulasjonsdeler.
http://answers.unity3d.com/questions/15250/how-to-declare-and-use-a-enum-variable-in-javascri.html

Videre kan dere vurdere å "higlight" den delen som er under manipulasjon. Til dette kan dere vurdere å bruke en uniform variabel for hvert delobjekt. Vurder å bruk en vec3 for å sette en fargeverdi(RGB). Denne uniformvariabelen må også shaderen ha.

Summary: It will be good enough to make a claw with two fingers, each having two parts.

Assignment 3 (No demonstration and no submission)

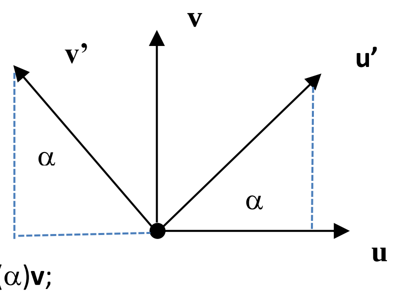
Camera part I

It is very important to study carefully the code of the project **MultiBodiesWithCamera** after reading the text below before doing assignments 4 and 5.

We want to be able to fly in a scene by moving the camera more specified in the text below and define some movements. In the camera system the eye is at the origin, the first axis is u-axis, the second axis is v-axis and the third axis is n-axis.
We can regard the camera as an object that can be moved.

From flight world (http://theboredengineers.com/2012/05/the-quadcopter-basics)
https://howthingsfly.si.edu/flight-dynamics
http://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html
http://webglfundamentals.org/webgl/webgl-3d-camera-look-at.html

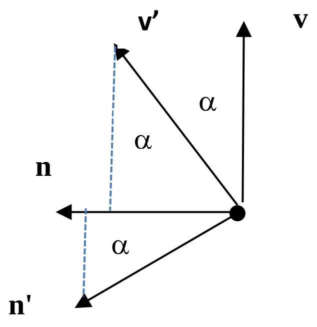
1. **Roll the camera**
   Rotate about n-axis



u' = cos(α)**u** + sin(α)**v**;
v' = -sin(α)**u** + cos(α)**v;**

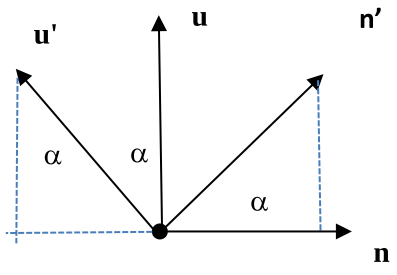
2. **Pitch the camera**
   Rotate about the u-axis

v' = cos(α)**v** + sin(α)**n**;
n' = -sin(α)**v** + cos(α)**n;**



3. **Yaw the camera**
   Rotate about the v-axis

u' = cos(α)**u** -sin(α)**n** ;
n' = sin(α)**u** + cos(α)**n;**



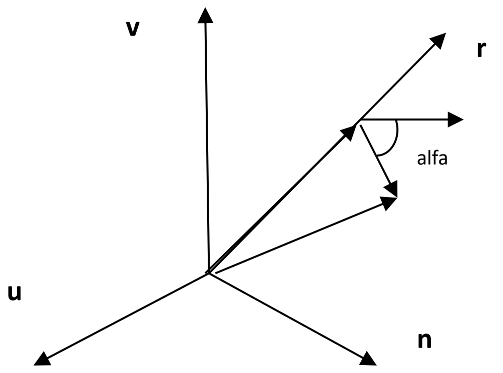
4. Slide (Two options)

Slide is moving the camera along one of the axes or combine several, but without rotating the camera. The camera looks along the negative n-axis. A movement along the n-axis is to move forward / backward. A movement along the u-axis is moving to the left / right and move along the v-axis is moving up/down.

To move a distance D along the u-axis is putting eye to eye + D$\vec{u}$. We can thus combine three possible upward slides in a function with parameters delU, delV and delN, meaning delU along $\vec{u}$, delV along $\vec{v}$ and delN along $\vec{n}$. You uppdate *eye*, but can also update look if you want. (Specify which option you want), but it's ok just to refresh the *eye*.

5. General rotation (Direct method without Euler)

You can use a method, rotate (Vector3 axis, float angle) rotating camera a given angle in degrees about an axis. This will rotate all three axes u, v and n on the eye.
The matrix representing this transformation is complex.



Ref: Real-Time Rendering: Tomas Akenine-Møller.

r is a vector defines the rotation axis. A strategy / method is to enter a local coordinate system with vectors: **r** vector defines the rotation axis.
A strategy / method is to enter a local coordinate system with vectors:

$\mathbf{r_z}' = \mathbf{r}$

$\mathbf{r_y}' = \mathbf{r} \times \mathbf{r_x} / |\mathbf{r} \times \mathbf{r_x}|$, cross product

$\mathbf{r_x}' = \mathbf{r_y}' \times \mathbf{r_z}'$

We then get the change matrix (norsk: overgangsmatrise) from new to old coordinates where the columns are the components of the new basis vectors (**u ', v', n** ') expressed by the old (**u, v, n**).Se lecture notes.

$$R_r(\alpha) == \begin{bmatrix} c+(1-c)r_x^2 & (1-c)r_yr_x - sr_z & (1-c)r_zr_x + sr_y & 0 \\ (1-c)r_xr_y + sr_z & c+(1-c)r_y^2 & (1-c)r_zr_y - sr_x & 0 \\ (1-c)r_xr_z - sr_y & (1-c)r_yr_z + sr_x & c+(1-c)r_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c = cos(α), og s = sin(α), α er rotation angle.

The components of the new basis vectors (**u '**, **v'** and **n '**) expressed by the old (**u, v**, and **n**) is given as columns in the matrix above. This then gives:

$$\mathbf{u}' = (c+(1-c)r_x^2)\mathbf{u} + ((1-c)r_y r_x + s r_z)\mathbf{v} + ((1-c)r_z r_x - s r_y)\mathbf{n}$$
$$\mathbf{v}' = ((1-c)r_x r_y - s r_z)\mathbf{u} + (c+(1-c)r_y^2)\mathbf{v} + ((1-c)r_z r_y + s r_x)\mathbf{n}$$
$$\mathbf{n}' = ((1-c)r_x r_z + s r_y)\mathbf{u} + ((1-c)r_y r_z - s r_x)\mathbf{v} + (c+(1-c)r_z^2)\mathbf{n}$$

In the MVnew.js : `rotate(angle, axis ){…}, lookAt( eye, at, up ){…}`

The change of basis matrix from world system (old) system) to camera system (new system) is given below.

$$\begin{bmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se lecture notes.This matrix is called the View matrix. The `lookAt(…)` - function is available in the math library. **This function creates and returns the View matrix**.
The Model -matrix is the composite matrix for all model transformations.
The View- matrix is multiplied by the Model matrix (V*M) and is called Model-View-matrix.
When we multiply Model-View matrix with coordinate for a vertex, we get the transformed vertex represented in the camera system.
Notation.

$$[dx, dy, dz] = [-\overrightarrow{eye}\cdot\vec{u}, -\overrightarrow{eye}\cdot\vec{v}, -\overrightarrow{eye}\cdot\vec{n}]$$

eye is here a position vector, (O - eye) from origin to the eye point in world system.

$$\overrightarrow{eye}\cdot\vec{u}; means; eye_x u_x + eye_y u_y + eye_z u_z$$

When changing the camera, the View-matrix has to be updated. Exercise: Study carefylly the code i camera.js.

```
From MVnew.js modified!function lookAt( eye, at, up ){
    …
  var n = normalize( subtract(eye, at) );    var u = normalize( cross(up, n) );   var v = normalize( cross(n, u) );
    var result = mat4(
  vec4( u, -dot(u, eye) ),     vec4( v, -dot(v, eye) ),      vec4( n, -dot(n, eye) ),    vec4()  );
  return result;

}
```

Assignment 4 (quaternion- no submission)

a) Run the project **trackball** in the folder **04**. The project realizes a virtual trackball. Study formulas for calculation of the new points and rotational axis (cross product is used to make a rotation axis). You will find explanations in the textbook 4.13 og 4.14.

b) What are the advantages of using quaternions? Run the project **cubeq** in the folder **04**. Study the code for rotation quaternion. Do the same for the project **trackballQuaternions**.

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
https://www.youtube.com/watch?v=zjMuIxRvygQ
https://www.3dgep.com/understanding-quaternions/
https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/

Assignment 5 (Demonstration and submission)

camera part II (using quaternion)

- En scenegraf er en struktur av noder som ofte er representert som en asyklisk rettet graf (DAG)
- Nodene representerer geometri, egenskaper og deres forhold til hverandre.
- Node (data og metoder) bl.a.
    - En forelder
    - Ett eller flere barn
    - Kan gjennomgå sine barn

Vi kommer mer tilbake til scenegraf senere.

An object defined in its object coordinate system.
The world matrix is the change matrix from object coordinate system to world coordinate system (transform vertices from obj. coordinate system to WC). The world matrix represents a combination of transformations. The different objects can have different world matrices.

The View-matrix is the change matrix form WC to VC (transform vertices from WC to VC).
This matrix is the reverse of camera's world matrix (translate and rotate a camera):
This follows by the rules for change of matrix, se lecture notes.

Hierarchical models: parent child relation.
We do local transformations when we doing transformations on the child in the child's system.
The transformations will be relative to the parent's system.
E.g. robot arm, recursive cubes (module Code).
General of concatenating affine transformations as S(scale), T(translate) and R(rotate).
E.g. : If we want first to scale, than rotate and finally translate.
The order will be:
$M = TRS$; $\mathbf{p}' = M\mathbf{p}$; $\mathbf{p}' = TRS\mathbf{p}$

If a node in the scene graph do not have a parent, than the world matrix (represents the world transformation) is the same as the local matrix (represents the local transformation).
$\text{Child}_{\text{world}} = \text{Child}_{\text{local}}$
General: We transform a child node by multiply by the parent's world matrix as:
$\text{Child}_{\text{world}}=\text{Parent}_{\text{world}}*\text{Child}_{\text{local}}$
Important: A scene graph have to update the world matrices for all the nodes.
Assignment: Download the project **dat155-camera-system-assignment** from the module Code.
Fill in the code missing in the files main.js and the file MouseLookController.js, explained in the comments-sentences such that the camera work properly. Remark: The project use this math-library: http://glmatrix.net/
You need to look at the documentation. Option: Expand by using a key to perform a roll motion of the camera.

Oppdatering av view-matrisen:
Bruker utvidelse av atferd i subklassen ved overkjøring av metoden tick

scene arver fra node og kamera arver fra node.
view-matrisen oppdateres i kamera-klassen.
node.tick() <- kamera.tick()
node.tick() <- scene.tick()