CS 580 – Discussion 4
HW 3
9/15/2015

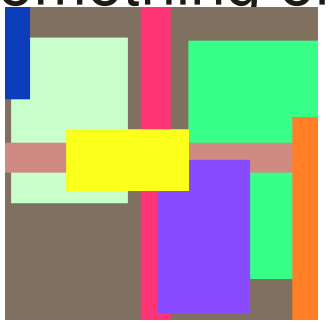
Matthias Hernandez

This week

- Questions on HW2 / Class material / Course organization?
- HW 3 description
- HW 3 pitfalls
- Q&A

So far

- HW 1: display something on the screen



- HW 2: rasterize triangles in screen coordinates

Goals of HW 3

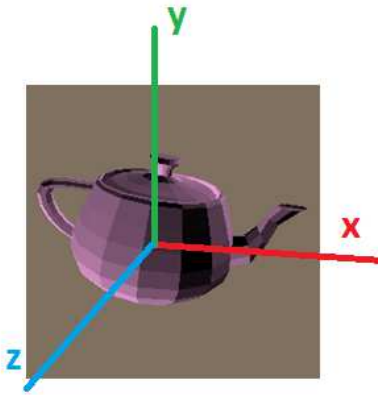
- 1. Change the viewpoint of the camera

Default camera

App-defined camera

Goals of HW 3

- 2. Enable transformations on the teapot



example: in app-defined camera


RotX = 30°


RotY = 30°


RotZ = 30°


Tr=[1 1 1]


Scale=[.5 .5 .5]

Setting up the code

- Open the CS580HW3.dsw file

- Copy disp.h and disp.cpp from your HW1.
- Add them to the Solution

- Copy rend.cpp from your HW2 –and rend.h if you modified it- into the rend.cpp.
  Warning: do not override the rend.cpp from HW3, it contains prototypes of the functions to fill out.

Goal 1: changing the camera

screen $\mathbf{X}_{sp}$ perspective (NDC) $\mathbf{X}_{pi}$ image $\mathbf{X}_{iw}$ world $\mathbf{X}_{wm}$ model

HW2

HW3

$\mathbf{X}_{wm}$ object positions (per frame or per instance)
$\mathbf{X}_{iw}$ camera position and orientation (per frame)
$\mathbf{X}_{pi}$ camera FOV (focal length or zoom) (per frame)
$\mathbf{X}_{sp}$ mapping NDC image to frame-buffer (per frame)

HW2: vertex coordinates were in screen coordinate.
HW3: vertex coordinates are in model coordinate.

Note: To go from model coordinate to screen coordinate, you need to multiply by $\mathbf{X}_{sp}\mathbf{X}_{pi}\mathbf{X}_{iw}\mathbf{X}_{wm}$.

Goal 1: changing the camera

screen $\mathbf{X}_{sp}$ perspective (NDC) $\mathbf{X}_{pi}$ image $\mathbf{X}_{iw}$ world $\mathbf{X}_{wm}$ model

$\mathbf{X}_{sp}$ mapping NDC image to frame-buffer (per frame)

Camera parameters:

$$\mathbf{X}_{sp} = \begin{pmatrix} xs/2 & 0 & 0 & xs/2 \\ 0 & -ys/2 & 0 & ys/2 \\ 0 & 0 & MAXINT & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[xs, ys] = [xres, yres]

[xres, yres]: size of the image

Goal 1: changing the camera

screen $\mathbf{X}_{sp}$ perspective (NDC) $\mathbf{X}_{pi}$ image $\mathbf{X}_{iw}$ world $\mathbf{X}_{wm}$ model

$\mathbf{X}_{pi}$ camera FOV (focal length or zoom) (per frame)

Camera parameters:
- FOV
- Xpi

$$\mathbf{X}_{pi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/d & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

$1/d = \tan(FOV / 2)$

FOV: field of view of the camera

Goal 1: changing the camera

screen $\mathbf{X}_{sp}$ perspective (NDC) $\mathbf{X}_{pi}$ image $\mathbf{X}_{iw}$ world $\mathbf{X}_{wm}$ model

$\mathbf{X}_{iw}$ camera position and orientation (per frame)

$$\mathbf{X}_{iw} = \begin{pmatrix} X_x & X_y & X_z & -X \bullet C \\ Y_x & Y_y & Y_z & -Y \bullet C \\ Z_x & Z_y & Z_z & -Z \bullet C \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Camera parameters:

- FOV
- Xpi
- C
- L
- Up
- Xiw

$Z = CL / || CL ||$
$Y = up' / || up' ||$
$X = (Y \times Z)$

where C camera origin, L look-at point
where up’ = up - (up • Z) Z

Goal 1: changing the camera

screen $\mathbf{X}_{sp}$ perspective (NDC) $\mathbf{X}_{pi}$ image $\mathbf{X}_{iw}$ world $\mathbf{X}_{wm}$ model

$\mathbf{X}_{wm}$ object positions (per frame or per instance)

Camera parameters:

- FOV
- Xpi
- C
- L
- Up
- Xiw

It is a concatenation of transformation matrices that will be applied to the model prior to rendering (Goal 2):

- **Translations**
- **Rotations**
- **Scaling**

Goal 1: changing the camera

- The camera structure is defined in gz.h.

```
typedef struct  GzCamera {
      GzMatrix      Xiw;          // Precomputed Xiw
      GzMatrix      Xpi;          // Precomputed Xpi
      GzCoord       position;     // C
      GzCoord       lookat;       // L
      GzCoord       worldup;      // up
      float         FOV;          // FOV
} GzCamera;
```

Goal 1: setting the camera

- GzPutCamera:
  - Set the camera parameters of render->camera
    - Position
    - Lookat
    - worldup
    - FOV
  - Compute Xpi, Xiw

Goal 1: What to do once you have computed Xsp, Xpi, Xiw and Xwm?

- Need to put the matrices on the stack: $X_{sp}X_{pi}X_{iw}X_{wm}$
- The stack contains the transformations that need to be applied to every vertex before display.

- In the code: the GzRender struct in rend.h now contains the camera and a stack of matrix transforms:

```
GzCamera camera;
short    matlevel;       /* top of stack - current xform */
GzMatrix Ximage[MATLEVELS]; /* stack of xforms (Xsm) */
```

Goal 1: Operations on the stack

- In rend.cpp, you will need to implement:
  - GzPushMatrix // Pushes a matrix on the stack
  - GzPopMatrix // Pops a matrix from the stack
- Then you will need to push *in that order*:
  - Xsp
  - Xpi
  - Xiw
  - Xwm

When initializing the display:
GzBeginRender

Goal 1: Operations on the stack

- GzPushMatrix:
  - If the stack is empty
    - Add the matrix
  - Otherwise
    - Multiply the new matrix by the top of the stack and push it into the stack
  - Increment matlevel

- GzPopMatrix:
  - Decrement matlevel

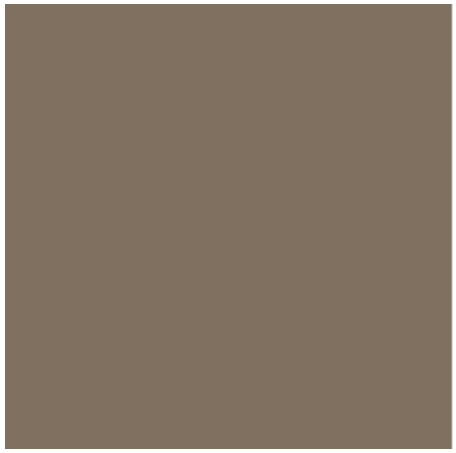
Goal 1: Using the transform stack

You will need to apply the set of transformations to every vertex of every triangle in **GzPutTriangle** **before rasterizing.**
Note: **Top of stack is** Ximage[matlevel]

*Warning***:** ignore triangles that are behind the view plane: skip any triangle with a negative screen-z vertex

Warning: Z-interpolation should be in screen space

Goal 1: result



Default

App-defined

Goal 1: Setting the default camera

You will need to set a default camera in GzNewRender. The parameters are in rend.h.

- Default lookat: [0 0 0]
- Default position: [DEFAULT_IM_X, DEFAULT_IM_Y, DEFAULT_IM_Z]
- Default worldup: [0 1 0]
- Default FOV: DEFAULT_FOV

Goal 1: result

Default

App-defined

To switch between default and app-defined cameras:
- in Application3.cpp, line 95.
#if 1: app-defined camera
#if 0: default camera

Goal 2: Adding transformations

Simply create the transformation matrices in rend.cpp:

- GzRotXMat // Rotation around X-axis
- GzRotYMat // Rotation around Y-axis
- GzRotZMat // Rotation around Z-axis
- GzTrxMat // Translation
- GzScaleMat // Scaling

- *Warning*: You need to convert the angles from degrees to radians: multiply by $\pi/180$

Goal 2: Rotations

$$R_x(\theta) \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) \Rightarrow \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) \Rightarrow \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Warning: Convert angles to radians:
multiply by $\pi/180$

Warning: Use homogeneous coordinates

Goal 2: Translations and Scaling

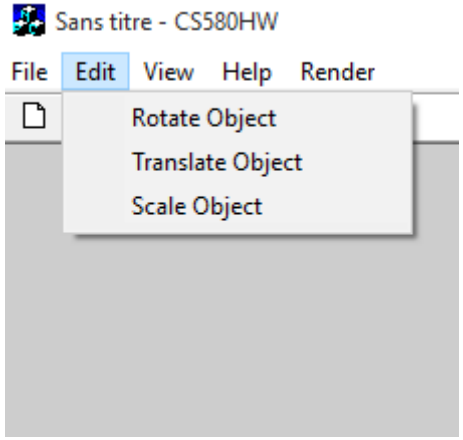
$$T(t_x, t_y, t_z)v \Rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ z' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

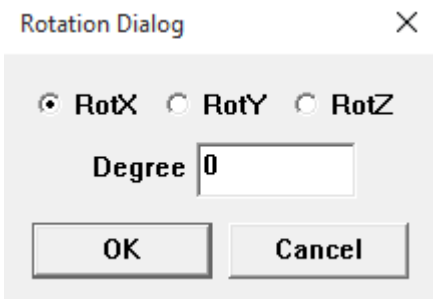
$$S(s_x, s_y, s_z)v \Rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ z' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \\ s_z \cdot z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

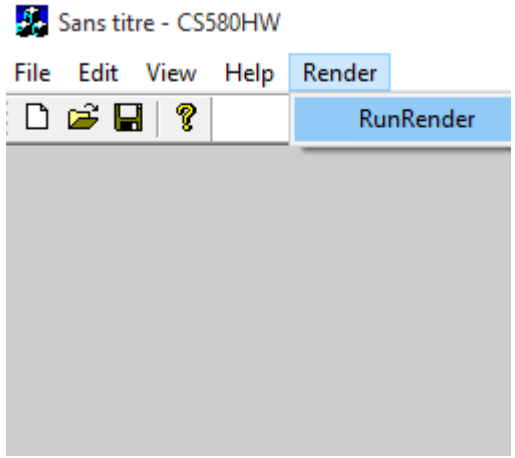
How to apply a transformation in the MFC

e.g. For a rotation of 30° around Y:

- Edit > Rotate Object > Rot Y > Enter 30 > OK
- Render > RunRender







HW3 pitfalls

- Do not forget to set default camera
- Careful when implementing dot, cross products and matrix multiplications
- Convert angles from degrees to radians
- Do not forget to apply the stack to every vertex before passing on to the rasterizer
- Use homogeneous coordinates: careful when converting 4-D to 3-D vectors.
  $[x\ y\ z\ w]^T \Rightarrow [x/w,\ y/w,\ z/w]^T$
- ignore triangles that are behind the view plane: skip any triangle with a negative screen-z vertex
- Z-interpolation should be in screen space

Q&A