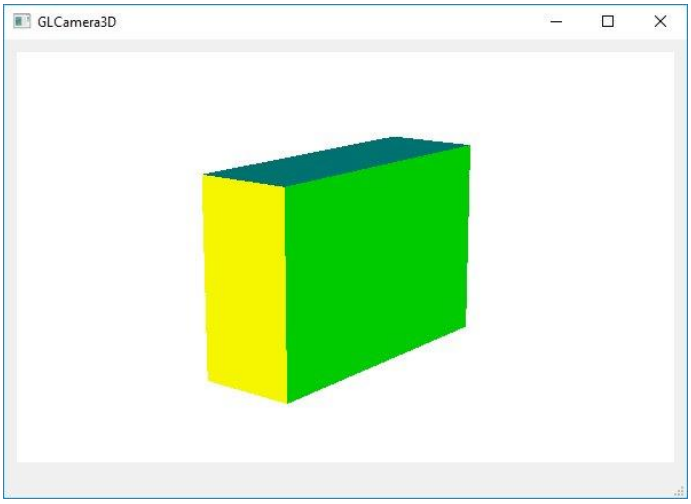
CIV 2802 – Sistemas Gráficos para Engenharia – PUC-Rio

Programa simples para manipulação de visualização 3D com o OpenGL

http://www.tecgraf.puc-rio.br/ftp_pub/lfm/CIV2802-Camera3D.zip



appmodel.py

```
from geometry.pnt3d import Pnt3D


class AppModel:

    def __init__(self):
        # Setup default object coordinates (a hexahedron)
        self.p0 = Pnt3D(0.0, 0.0, 0.0)
        self.p1 = Pnt3D(1.0, 0.0, 0.0)
        self.p2 = Pnt3D(1.0, 2.0, 0.0)
        self.p3 = Pnt3D(0.0, 2.0, 0.0)
        self.p4 = Pnt3D(0.0, 0.0, 3.0)
        self.p5 = Pnt3D(1.0, 0.0, 3.0)
        self.p6 = Pnt3D(1.0, 2.0, 3.0)
        self.p7 = Pnt3D(0.0, 2.0, 3.0)

        # Faces: -X, -Y, -Z, +X, +Y, +Z
        self.faces = [[self.p0, self.p4, self.p7, self.p3], \
                      [self.p0, self.p1, self.p5, self.p4], \
                      [self.p3, self.p2, self.p1, self.p0], \
                      [self.p1, self.p2, self.p6, self.p5], \
                      [self.p2, self.p3, self.p7, self.p6], \
                      [self.p4, self.p5, self.p6, self.p7]]
        self.fnormals = [[[-1.0,0.0,0.0], [-1.0,0.0,0.0], [-1.0,0.0,0.0], [-1.0,0.0,0.0]],\
                         [[ 0.0,-1.0,0.0], [ 0.0,-1.0,0.0], [ 0.0,-1.0,0.0], [0.0,-1.0,0.0]],\
                         [[ 0.0,0.0,-1.0], [ 0.0,0.0,-1.0], [ 0.0,0.0,-1.0], [0.0,0.0,-1.0]],\
                         [[ 1.0,0.0,0.0], [1.0,0.0,0.0], [1.0,0.0,0.0], [1.0,0.0,0.0]],\
                         [[ 0.0,1.0,0.0], [ 0.0,1.0,0.0], [0.0,1.0,0.0], [0.0,1.0, 0.0]],\
                         [[ 0.0,0.0,1.0], [ 0.0,0.0,1.0], [0.0,0.0,1.0], [0.0,0.0,1.0]]]
        # Colors: green, cyan, yellow, green, cyan, yellow
        self.fcolors = [[[0.0,1.0,0.0], [0.0,1.0,0.0], [0.0,1.0,0.0], [0.0,1.0,0.0]],\
                        [[0.0,1.0,1.0], [0.0,1.0,1.0], [0.0,1.0,1.0], [0.0,1.0,1.0]],\
                        [[1.0,1.0,0.0], [1.0,1.0,0.0], [1.0,1.0,0.0], [1.0,1.0,0.0]],\
                        [[0.0,1.0,0.0], [0.0,1.0,0.0], [0.0,1.0,0.0], [0.0,1.0,0.0]],\
                        [[0.0,1.0,1.0], [0.0,1.0,1.0], [0.0,1.0,1.0], [0.0,1.0,1.0]], \
                        [[1.0,1.0,0.0], [1.0,1.0,0.0], [1.0,1.0,0.0], [1.0,1.0,0.0]]]

    def isEmpty(self):
        return self.faces == []
```

```
    def getFaces(self):
        return self.faces

    def getFacePts(self, _faceID):
        pts = self.faces[_faceID]
        return pts

    def getFaceNormals(self, _faceID):
        fnormal = self.fnormals[_faceID]
        return fnormal

    def getFaceColors(self, _faceID):
        fcolor = self.fcolors[_faceID]
        return fcolor
```

glcanvas.py

```
from PyQt5 import QtOpenGL
from OpenGL.GL import *
from OpenGL.GLU import *
import math


class GLCanvas(QtOpenGL.QGLWidget):

    def __init__(self, _controller=[], _view=[]):
        super(GLCanvas, self).__init__()

        # canvas sizes
        self.width = 0   # width of canvas (horizontal raster size)
        self.height = 0  # height of canvas (vertical raster size)

        # camera (eye) parameters
        self.eyex = 0.0    # camera (eye) x coordinate in object space
        self.eyey = 0.0    # camera (eye) y coordinate in object space
        self.eyez = 1.0    # camera (eye) z coordinate in object space
        self.refx = 0.0    # reference point x coordinate in object space
        self.refy = 0.0    # reference point y coordinate in object space
        self.refz = 0.0    # reference point z coordinate in object space
        self.vupx = 0.0    # view up-vector x component in object space
        self.vupy = 0.0    # view up-vector y component in object space
        self.vupz = 1.0    # view up-vector z component in object space

        # model clipping volume limits
        self.left = -1.0    # left limit of clipping volume (eye space)
        self.right = 1.0    # right limit of clipping volume (eye space)
        self.bottom = -1.0  # bottom limit of clipping volume (eye space)
        self.top = 1.0      # top limit of clipping volume (eye space)
        self.front = 1.0    # front limit of clipping volume (eye space)
        self.back = 2.0     # back limit of clipping volume (eye space)

        # model rotation parameters
        self.frames = 360  # number of frames in a complete 360 degree rotation
        self.alpha = 0.0   # rotation angle
        self.delta = 2.0 * math.pi / float(self.frames)  # rotation increment angle

        # canvas aggregation objects
        self.controller = _controller  # handle to app controller
        self.view = _view  # handle to app view

        # OpenGL display lists and display update flags
        self.viewDsp = 0  # GL list index for model display
        self.updatedDsp = False  # if true, model display is updated

    # ------------------------------------------------------------------
    # ------------------------------------------------------------------
```

```
# --------------------GENERAL PROPERTY METHODS-----------------------
# -------------------------------------------------------------------
# -------------------------------------------------------------------

# -------------------------------------------------------------------
def setController(self, _controller):
    self.controller = _controller

# -------------------------------------------------------------------
def getController(self):
    return self.controller

# -------------------------------------------------------------------
def setView(self, _view):
    self.view = _view

# -------------------------------------------------------------------
def getView(self):
    return self.view

# -------------------------------------------------------------------
def resetViewDisplay(self):
    if self.view is None:
        return
    self.updatedDsp = False
    self.update()

# -------------------------------------------------------------------
# --------------------CANVAS PREDEFINED SLOTS------------------------
# -------------------------------------------------------------------
# -------------------------------------------------------------------

# -------------------------------------------------------------------
def initializeGL(self):
    # Set lighting and depth test parameters
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE)
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE)
    glEnable(GL_COLOR_MATERIAL)
    glEnable(GL_CULL_FACE)
    glDisable(GL_DEPTH_TEST)

    # set canvas background color
    color = self.view.getBackgroundColor()
    glClearColor(color[0], color[1], color[2], 1.0)
    glClear(GL_COLOR_BUFFER_BIT)

# -------------------------------------------------------------------
def resizeGL(self, _width, _height):

    # Avoid division by zero
    if _width == 0:
        _width = 1

    # store GL canvas sizes in object properties
    self.width = _width
    self.height = _height

    # Setup visualization parameters based on model bounding box
    if (self.view is not None) and (not self.view.isEmpty()):
        self.resetView()
        self.alpha = 0.0;  # reset rotation angle

    # Setup the viewport to canvas dimesions
    glViewport(0, 0, self.width, self.height)
```

```
    # Setup (clipping) view volume and perpective projection
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glFrustum(self.left,self.right,self.bottom,self.top,self.front,self.back)

# ------------------------------------------------------------------
def paintGL(self):

    # clear the buffer with the current color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    # Check to see whether there is a model
    if (self.view is None) or self.view.isEmpty():
        return

    # Setup default head light position (before applying camara
    # transformation) and setup camera parameters
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    light_pos = [0.0, 0.0, 1.0, 0.0]
    glLightfv( GL_LIGHT0, GL_POSITION, light_pos )
    gluLookAt( self.eyex, self.eyey, self.eyez, \
               self.refx, self.refy, self.refz, \
               self.vupx, self.vupy, self.vupz )

    # Display view , storing it in a display list
    if not self.updatedDsp:
        if self.viewDsp > 0:
            glDeleteLists(self.viewDsp, 1)
        self.viewDsp = self.makeDisplayView()

    if self.viewDsp > 0:
        glCallList(self.viewDsp)
        self.updatedDsp = True

# ------------------------------------------------------------------
# ------------------------------------------------------------------
# ----------------------DISPLAY FUNCTIONS---------------------------
# ------------------------------------------------------------------
# ------------------------------------------------------------------

# ------------------------------------------------------------------
def drawFaces(self):
    faces = self.view.getFaces()
    for i in range(0, len(faces)):
        fcolors = self.view.getFaceColors(i)
        fnormals = self.view.getFaceNormals(i)
        pts = self.view.getFacePts(i)
        if len(pts) == 3:
            glBegin(GL_TRIANGLES)
        elif len(pts) == 4:
            glBegin(GL_QUADS)
        else:
            glBegin(GL_POLYGON)
        for j in range(len(pts)):
            glColor3f(fcolors[j][0], fcolors[j][1], fcolors[j][2])
            glNormal3f(fnormals[j][0], fnormals[j][1], fnormals[j][2])
            glVertex3f(pts[j].getX(), pts[j].getY(), pts[j].getZ())
        glEnd()

# ------------------------------------------------------------------
def makeDisplayView(self):
    if (self.view is None) or (self.view.isEmpty()):
        return 0
```

```
    list = glGenLists(1)
    glNewList(list, GL_COMPILE)
    self.drawFaces()
    glEndList()
    return list

# ---------------------------------------------------------------------
# ---------------------------------------------------------------------
# ------------ METHODS TO MANAGE VISUALIZATION PARAMETERS -------------
# ---------------------------------------------------------------------
# ---------------------------------------------------------------------

# ---------------------------------------------------------------------
def resetView(self):
    if (self.view is None) or self.view.isEmpty():
        return

    # Get model bounding box
    xmin, xmax, ymin, ymax, zmin, zmax = self.view.getBoundBox()

    # Get model max box size
    sizex = xmax - xmin
    sizey = ymax - ymin
    sizez = zmax - zmin
    if sizex > sizey:
        max_size = sizex
    else:
        max_size = sizey
    if max_size < sizez:
        max_size = sizez

    # Place reference point at center of bounding box.
    # **** COMPLETE HERE - GLCANVAS: 01 ****



    # **** COMPLETE HERE - GLCANVAS: 01 ****

    # Place camera with same x coordinate of reference point, distant
    # 1 "max_size" to reference point in y direction, and distant
    # 4 "max_sizes" to reference point in z direction
    # **** COMPLETE HERE - GLCANVAS: 02 ****



    # **** COMPLETE HERE - GLCANVAS: 02 ****

    #  Define camera vertical plane perpendicular to xz plane.
    # **** COMPLETE HERE - GLCANVAS: 03 ****



    # **** COMPLETE HERE - GLCANVAS: 03 ****

    # Compute canvas viewport aspect ratio.
    vpr = float(self.height) / float(self.width)

    # Define view window with minimum size equal to model
    # maximum size and with aspect ratio equal to canvas.
    view_window_w = view_window_h = max_size
    # **** COMPLETE HERE - GLCANVAS: 04 ****



    # **** COMPLETE HERE - GLCANVAS: 04 ****
```

```
        # Define view volume parameters such that:
        # (1) view window center coincides with canvas center
        # (2) front clip plane is located at 3 "max_sizes" from camera
        # (3) back  clip plane is located at 5 "max_sizes" from camera
        # **** COMPLETE HERE - GLCANVAS: 05 ****




        # **** COMPLETE HERE - GLCANVAS: 05 ****

    # -----------------------------------------------------------------------
    def rotateView(self, _increase):

        if (self.view is None) or self.view.isEmpty():
            return

        # Increment or decrement rotation angle
        if _increase:
            self.alpha += self.delta
        else:
            self.alpha -= self.delta

        # Define rotation radius equal to distance between camera and
        # reference point projected on xz plane.
        radius = math.sqrt((self.eyex - self.refx) * (self.eyex - self.refx) + \
                           (self.eyez - self.refz) * (self.eyez - self.refz))

        # Update camera (eye) position, applying a rotation about
        # reference point around y axis
        # **** COMPLETE HERE - GLCANVAS: 06 ****


        # **** COMPLETE HERE - GLCANVAS: 06 ****

        self.update()

    # -----------------------------------------------------------------------
    # -----------------------------------------------------------------------
    # ----------------------MOUSE EVENT SLOTS--------------------------------
    # -----------------------------------------------------------------------
    # -----------------------------------------------------------------------

    # -----------------------------------------------------------------------
    def wheelEvent(self, event):
        if event.angleDelta().y() > 0:
            increase = True
        else:
            increase = False

        self.rotateView(increase)
```