3D Projective Viewing with House

Ross Beveridge, October 23, 2018

It is helpful to be able to review the linear algebra and the 3D geometry of viewing using a tool such as Sage. The example setup here originated as a series of Maple Worksheets I developed in CS 410 nearly 20 years ago. Here the example is updated and this example carries two key ideas. You should see the mathematics of the four major components of the Perspective Projection Pipeline geometry in both symbolic form and also in specific numerical examples. Second, the object moddel, a house, can be viewed using a 3D viewer both before and after being transformed in a camera's canoncial view volume.

Keep in mind the canonical view volume is a cube bounded between -1 and 1 along the camera's X, Y and Z axes. The X and Y axes are the horizontal and vertical axes of the image plane. The Z axes carries pseudo-depth information. The 'psuedo' is appened to the front of 'depth' to highlight that while larger values mean further from the camera, a non-linear warping is introduced. This can be seen in the examples below.

This one Notebook includes both symbolic forms of the key components in the transformation along with different concrete examples. There are a series of different examples supported in this worksheet. These are defined and selected between below. First, some basic Sage graphics defaults are customized and then the house model is defined and viewed.

In [1]:
```
from sage.plot.plot3d.base import SHOW_DEFAULTS
SHOW_DEFAULTS['aspect_ratio'] = (1,1,1)
SHOW_DEFAULTS['frame_aspect_ratio'] = (1,1,1)
SHOW_DEFAULTS['perspective_depth'] = false
```

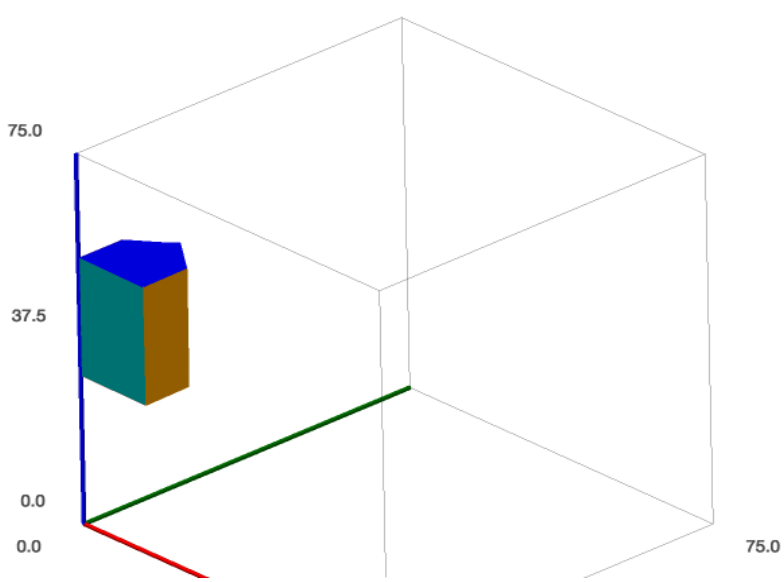
Construct the house model. This means a list of lists to define the vertices along with a 4 by 10 matrix of vertices in homogeneous coordinates wtih one vertex per column. Finally, the faces are named in a dictionary that includes both a color to draw the face and a sequence of face vertex indices.

In [2]:
```
VVH1 = [[0,0,30],[0,10,30],[8,16,30],[16,10,30],[16,0,30],[0,0,54],[0,10,54],[8,16,54],[16,10,54],[16,0,54]]
MMH1 = matrix( len(VVH1), 3, VVH1).augment(matrix(ZZ, len(VVH1), 1, lambda x,y: 1))
MMH1 = MMH1.transpose()
faceV   = {
'houseFront' : ((0,1,2,3,4),'tomato'),     'houseBack' : ((5,6,7,8,9), 'blue'),
'wallRight'   : ((0,1,6,5),  'brown'),      'wallLeft' : ((4,3,8,9),   'orange'),
'roofRight'   : ((1,2,7,6),  'greenyellow'),'roofLeft' : ((3,2,7,8),   'springgreen'),
'floor'      : ((0,4,9,5),  'darkcyan')}
MMH1
```

Out[2]:
```
[ 0  0  8 16 16  0  0  8 16 16]
[ 0 10 16 10  0  0 10 16 10  0]
[30 30 30 30 30 54 54 54 54 54]
[ 1  1  1  1  1  1  1  1  1  1]
```

Create a 3D view of the house that will facilitate your understanding of how to place camera relative to the house geometry.

In [3]:
```
def gHouseFace(vis) :
    vrts = [[MMH1[i,j] for i in range(3)] for j in vis[0]]
    return polygon3d(vrts,color=vis[1],alpha=1.0)
def gAxe(pt,c) :
    return line3d([(0,0,0),pt],thickness=5,color=c)
def axes() :
    bnd = 75
    return map(gAxe,[(bnd,0,0),(0,bnd,0),(0,0,bnd)],['red','green','blue'])
figcon1 = sum(map(gHouseFace,faceV.values()) + axes())
figcon1.show()
```



A series of different examples are rolled up in this one notebook. They are defined below using a python Dictionary.

The examples currently implemented are:

'sym' Generate the camera to canonical view volume transformation in symbolic terms and display the pipeline of four matrices

'ex1' Place the camera out beyond the house along the positive Z (world) axis. Then move the camera out the X and Y axis to get a centered view of the house's back face.

'ex2' Move the camera 'to the right' and also set the far clipping plane significantly further back. Note the non-linear treatment of the pseudo-depth captured by Z in camera coordinates.

'ex3' Place the camera to get a view of both the back side from a bit above with significant perspective.

'ex4' A placement very like the previous, but with the camera moved far from the object and the bounds tightened to gain a similar size view but with little perspective.

In [4]:
```
var('umin, umax, vmin, vmax, near, far')
casecams = {
   'sym' : {
       'eye'  : vector(SR, 3, var('ex, ey, ez')),
       'look' : vector(SR, 3, var('lx, ly, lz')),
       'up'   : vector(SR, 3, var('upx, upy, upz')),
       'bnds' : { 'left' : umin, 'right' : umax, 'bottom' : vmin, 'top' : vmax},
       'nefa' : { 'near' : near, 'far' : far}
   },
   'ex1' : {
       'eye'  : vector(SR, 3, (8, 8, 100)),
       'look' : vector(SR, 3, (8, 8, 54)),
       'up'   : vector(SR, 3, (0, 1, 0)),
       'bnds' : { 'left' :-20, 'right' : 20, 'bottom' : -20, 'top' : 20},
       'nefa' : { 'near' : -45, 'far' : -75}
   },
   'ex2' : {
       'eye'  : vector(SR, 3, (40, 8, 100)),
       'look' : vector(SR, 3, (8, 8, 42)),
       'up'   : vector(SR, 3, (0, 1, 0)),
       'bnds' : { 'left' :-15, 'right' : 15, 'bottom' : -15, 'top' : 15},
       'nefa' : { 'near' : -50, 'far' : -1000}
   },
   'ex3' : {
       'eye'  : vector(SR, 3, (20, 20, 60)),
       'look' : vector(SR, 3, (8, 8, 42)),
       'up'   : vector(SR, 3, (0, 1, 0)),
       'bnds' : { 'left' :-15, 'right' : 15, 'bottom' : -15, 'top' : 15},
       'nefa' : { 'near' : -10, 'far' : -100}
   },
   'ex4' : {
       'eye'  : vector(SR, 3, (128, 128, 222)),
       'look' : vector(SR, 3, (8, 8, 42)),
       'up'   : vector(SR, 3, (0, 1, 0)),
       'bnds' : { 'left' : -12, 'right' : 12, 'bottom' : -12, 'top' : 12},
       'nefa' : { 'near' : -200, 'far' : -300}
   }
}
case = 'ex2'
```

What follows is a complete illustration of how to construct the transformation matrices - ultimately a single matrix - that accomplishes 3D project of a model in the world in to a camera's canonical view volume.

The transformation consists of four parts.

1. Shift the origin to the eye position
2. Rotate so that z-axis aligns with eye minus lookat and up vector defines camera y-axis
3. Apply prespective projection to place 3D points into canoncal view volume, introducing perspective effect
4. Normalize camera X, Y and Z so to match standard -1 to 1 for X and Y and 0 to 1 bounds on cannonical view volume

In [5]:
```
cam  = casecams[case]  # Choose a symbolic or numeric example
TM = matrix.identity(SR, 4); TM[0:3,3] = -cam['eye']
EV = cam['eye'] ; LV = cam['look']; UP = cam['up']
# The next three lines are critical, they establish camera orientation
WV = EV - LV; WV = WV / WV.norm();
UV = UP.cross_product(WV); UV = UV / UV.norm();
VV = WV.cross_product(UV);
R3 = matrix(SR, 3,3, (UV, VV, WV));
RM = matrix.identity(SR, 4); RM[0:3,0:3] = R3;
OM = matrix.identity(SR, 4);
OM[0,0] = 2 / (cam['bnds']['right'] - cam['bnds']['left'])
OM[1,1] = 2 / (cam['bnds']['top']   - cam['bnds']['bottom'])
OM[2,2] = 2 / (cam['nefa']['near']  - cam['nefa']['far'])
OM[0,3] = - ((cam['bnds']['right'] + cam['bnds']['left'])   / (cam['bnds']['right'] - cam['bnds']['left']))
OM[1,3] = - ((cam['bnds']['top']   + cam['bnds']['bottom']) / (cam['bnds']['top']   - cam['bnds']['bottom']))
OM[2,3] = - ((cam['nefa']['near']  + cam['nefa']['far'])    / (cam['nefa']['near']  - cam['nefa']['far']))
PM = matrix(SR, 4,4, 0)
PM[0,0] = cam['nefa']['near']
PM[1,1] = cam['nefa']['near']
PM[2,2] = cam['nefa']['near'] + cam['nefa']['far']
PM[2,3] = - cam['nefa']['near'] * cam['nefa']['far']
PM[3,2] = 1
WCM = OM * PM * RM * TM;
if (case == 'sym') :
    pretty_print(LatexExpr("{M_{WC} \: = \: M_{O} \: M_{P} \: M_{R} \: M_{T}}"))
    pretty_print(LatexExpr("{M_{T} \: = \:}"), TM)
    print("Symbolic form or Rotation not printed")
    # pretty_print(LatexExpr("{M_{R} \: = \:}"), RM)
    pretty_print(LatexExpr("{M_{P} \: = \:}"), PM)
    pretty_print(LatexExpr("{M_{O} \: = \:}"), OM)
if (case != 'sym') :
    pretty_print(LatexExpr("{M_{WC} \: = \: M_{O} \: M_{P} \: M_{R} \: M_{T}}"))
    pretty_print(LatexExpr("{M_{T} \: = \:}"), TM)
    pretty_print(LatexExpr("{M_{R} \: = \:}"), RM)
    pretty_print(LatexExpr("{M_{P} \: = \:}"), PM)
    pretty_print(LatexExpr("{M_{O} \: = \:}"), OM)
    pretty_print(LatexExpr("{M_{WC} \: = \:}"), WCM)
```

$$M_{WC} \; = \; M_{O} \; M_{P} \; M_{R} \; M_{T}$$

$$M_{T} \; = \; \begin{pmatrix} 1 & 0 & 0 & -40 \\ 0 & 1 & 0 & -8 \\ 0 & 0 & 1 & -100 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{R} \; = \; \begin{pmatrix} \frac{29}{1097}\sqrt{1097} & 0 & -\frac{16}{1097}\sqrt{1097} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{16}{1097}\sqrt{1097} & 0 & \frac{29}{1097}\sqrt{1097} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{P} \; = \; \begin{pmatrix} -50 & 0 & 0 & 0 \\ 0 & -50 & 0 & 0 \\ 0 & 0 & -1050 & -50000 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$M_{O} \; = \; \begin{pmatrix} \frac{1}{15} & 0 & 0 & 0 \\ 0 & \frac{1}{15} & 0 & 0 \\ 0 & 0 & \frac{1}{475} & \frac{21}{19} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

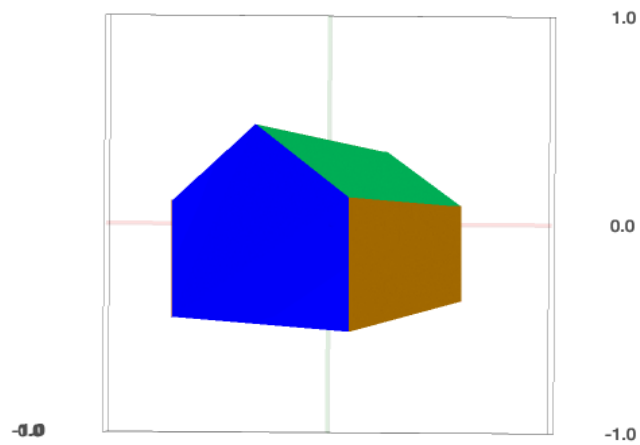
$$M_{WC} \; = \; \begin{pmatrix} -\frac{290}{3291}\sqrt{1097} & 0 & \frac{160}{3291}\sqrt{1097} & -\frac{4400}{3291}\sqrt{1097} \\ 0 & -\frac{10}{3} & 0 & \frac{80}{3} \\ -\frac{336}{20843}\sqrt{1097} & 0 & -\frac{609}{20843}\sqrt{1097} & \frac{74340}{20843}\sqrt{1097} - \frac{2000}{19} \\ \frac{16}{1097}\sqrt{1097} & 0 & \frac{29}{1097}\sqrt{1097} & -\frac{3540}{1097}\sqrt{1097} \end{pmatrix}$$

Once the single matrix is constructed, provided a numerical case is being run, the code below transforms vertices from world coordinates into camera coordinates.

In [6]:
```
def can_points(M) :
    for j in range(M.ncols()):
       for i in [0..3]:
          M[i,j] = M[i,j]/M[3,j];
if (case != 'sym') :
    MMH2 = WCM * MMH1
    can_points(MMH2)
    MMH2 = MMH2.apply_map(RR)
    pretty_print(MMH2.n(digits=2))
```

$$\begin{pmatrix} -0.050 & -0.050 & 0.25 & 0.59 & 0.59 & -0.72 & -0.72 & -0.35 & 0.078 & 0.078 \\ -0.33 & 0.083 & 0.35 & 0.091 & -0.37 & -0.45 & 0.11 & 0.48 & 0.13 & -0.51 \\ 0.20 & 0.20 & 0.27 & 0.34 & 0.34 & 0.66 & 0.66 & 0.78 & 0.92 & 0.92 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{pmatrix}$$

In [8]:
```
if (case != 'sym') :
    def gHouseFace(vis) :
        vrts = [[MMH2[i,j] for i in range(3)] for j in vis[0]]
        return polygon3d(vrts,color=vis[1],alpha=1.0)
    def axe2(pt,c) :
        return line3d([-pt,pt],thickness=5,color=c,opacity=0.1)
    def axes2() :
        return map(axe2,[vector([1,0,0]),vector([0,1,0]),vector([0,0,1])],['red','green','blue'])
    figcon2 = sum(map(gHouseFace,faceV.values()) + axes2())
    figcon2.show()
```



In [ ]: