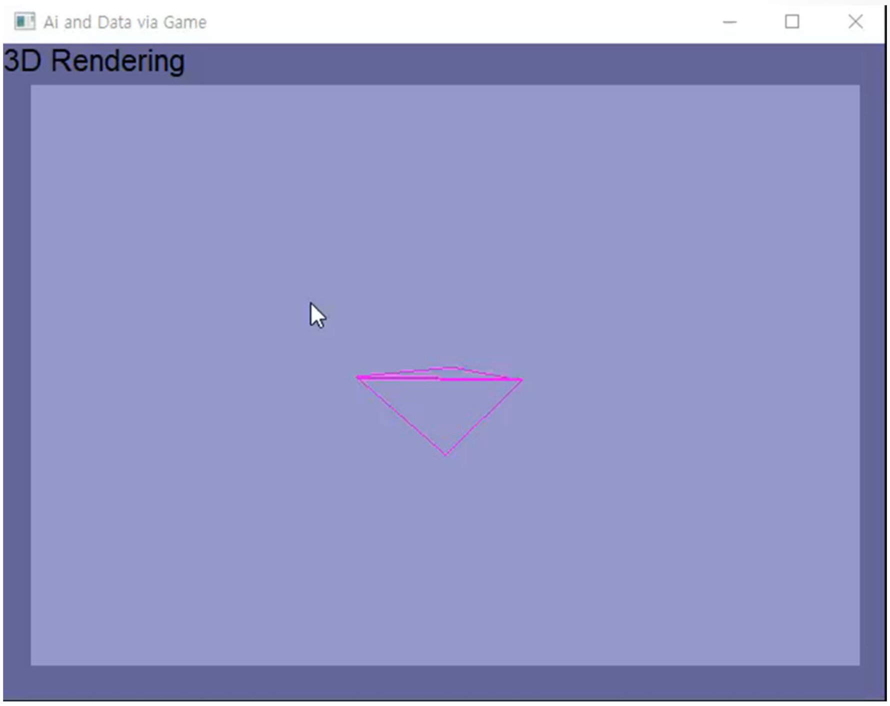
4. 3D Rendering



KhuGleBase.h (1)

```
class CKgVector3D {
public:
  double x, y, z;

  CKgVector3D() : x(0.), y(0.), z(0.) {}
  CKgVector3D(double xx, double yy, double zz) :
        x(xx), y(yy), z(zz) {}

  static double abs(CKgVector3D v);
  void Normalize();
  double Dot(CKgVector3D v1);
  CKgVector3D Cross(CKgVector3D v);
  CKgVector3D operator+ (CKgVector3D v);
  CKgVector3D operator- (CKgVector3D v);
  CKgVector3D operator- ();
  CKgVector3D &operator+= (CKgVector3D v);
};
CKgVector3D operator*(double s, CKgVector3D v);
```


KhuGleBase.cpp (1)

```
double CKgVector3D::abs(CKgVector3D v) {
  return sqrt(v.x*v.x + v.y*v.y + v.z*v.z);
}

void CKgVector3D::Normalize() {
  double Magnitude = abs(*this);
  if(Magnitude == 0) return;
  x /= Magnitude;        y /= Magnitude;           z /= Magnitude;
}

double CKgVector3D::Dot(CKgVector3D v) {
  return x*v.x + y*v.y + z*v.z;
}

CKgVector3D CKgVector3D::Cross(CKgVector3D v) {
  CKgVector3D NewV;
  NewV.x = y*v.z - z*v.y;
  NewV.y = z*v.x - x*v.z;
  NewV.z = x*v.y - y*v.x;
  return NewV;
}
```

this * V

$$\mathbf{a}\times\mathbf{b} = \begin{vmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

KhuGleBase.cpp (2)

```
CKgVector3D CKgVector3D::operator+ (CKgVector3D v) {
  return CKgVector3D(x+v.x, y+v.y, z+v.z);
}

CKgVector3D CKgVector3D::operator- (CKgVector3D v) {
  return CKgVector3D(x-v.x, y-v.y, z-v.z);
}

CKgVector3D CKgVector3D::operator- () {
  return CKgVector3D(-x, -y, -z);
}

CKgVector3D &CKgVector3D::operator+= (CKgVector3D v) {
  *this = *this + v;
  return *this;
}

CKgVector3D operator*(double s, CKgVector3D v) {
  return CKgVector3D(s*v.x, s*v.y, s*v.z);
}
```


Inverse matrix (1)

- Lower-upper decomposition, factorization
  - lower triangular matrix/upper triangular matrix

$$\mathbf{A} = \mathbf{LU}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \alpha_{21} & 1 & 0 & 0 \\ \alpha_{31} & \alpha_{32} & 1 & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & 1 \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix}$$

$$= \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ \alpha_{21}\beta_{11} & \alpha_{21}\beta_{12} + \beta_{22} & \alpha_{21}\beta_{13} + \beta_{23} & \ldots \\ \alpha_{31}\beta_{11} & \alpha_{31}\beta_{12} + \alpha_{32}\beta_{22} & \alpha_{31}\beta_{13} + \alpha_{32}\beta_{23} + \beta_{33} & \ldots \\ \alpha_{41}\beta_{11} & \alpha_{41}\beta_{12} + \alpha_{42}\beta_{22} & \ldots & \ldots \end{pmatrix}$$

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right)$$

Forward/back substitution

Inverse matrix (2)

$$\mathbf{Ax} = \mathbf{b}$$
$$\mathbf{LUx} = \mathbf{b}$$
$$\mathbf{Ly} = \mathbf{b}$$
$$\mathbf{Ux} = \mathbf{y}$$

$$\mathbf{AB} = \mathbf{I}$$
$$\mathbf{Ab}_1 = \mathbf{e}_1$$
$$\mathbf{Ab}_n = \mathbf{e}_n$$

$$\mathbf{Ab}_1 = \mathbf{e}_1$$
$$\mathbf{LUb}_1 = \mathbf{e}_1 \qquad \leftarrow \mathbf{z} = \mathbf{Ub}_1$$

$\mathbf{Lz} = \mathbf{e}_1$

Forward substitution

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ \alpha_{21} & 1 & 0 & 0 \\ \alpha_{31} & \alpha_{32} & 1 & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$\mathbf{Ub}_1 = \mathbf{z}$

Back substitution

$$\begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

InverseMatrix: KhuGleBase.h/cpp

Inverse matrix (3)

```
bool InverseMatrix(double  **a, double **y, int nN) {
  double **CopyA = dmatrix(nN, nN);
  double *col = new double[nN];
  int *indx = new int[nN];
  double d;

  for(int r = 0; r < nN; r++)
    for(int c = 0; c < nN; c++)
         CopyA[r][c] = a[r][c];

  if(!ludcmp(CopyA, nN, indx, &d)) {
    free_dmatrix(CopyA, nN, nN);

    delete[] indx;
    delete[] col;
    return false;
  }
```

Inverse matrix (4)

```
  for(int j = 0; j < nN; j++) {
    for(int i = 0; i < nN; i++)
        col[i] = 0.0;
    col[j] = 1.0;
    lubksb(CopyA, nN, indx, col);
    for(int i = 0; i < nN; i++)
        y[i][j] = col[i];
  }

  free_dmatrix(CopyA, nN, nN);

  delete[] indx;
  delete[] col;

  return true;
}
```

e_n 구하기

lu back sub

LU decomposition

Inverse matrix (5)

```
bool ludcmp(double **a, int nN, int *indx, double *d) {
  int i, imax, j, k;
  double big, dum, sum, temp;
  double *vv = new double[nN];
  const double TinyValue = 1.0e-20;

  *d = 1.0;
  for(i = 0; i < nN; i++) {
    big = 0.0;
    for (j = 0; j < nN; j++)
        if ((temp = fabs(a[i][j])) > big) big = temp;
    if (big == 0.0) {
        delete[] vv;    return false; // Singular
    }
    vv[i] = 1.0 / big;
  }
```

Inverse matrix (6)

```
for(j = 0; j < nN; j++) {
  for(i = 0; i < j; i++) {
       sum = a[i][j];
       for(k = 0; k < i; k++)  sum -= a[i][k] * a[k][j];
       a[i][j] = sum;
  }

  big = 0.0;
  for(i = j; i < nN; i++) {
       sum = a[i][j];
       for (k = 0; k < j; k++)
         sum -= a[i][k] * a[k][j];
       a[i][j] = sum;
       if ((dum = vv[i] * fabs(sum)) >= big) {
         big = dum; imax = i;
       }
  }
```

$$\mathbf{PA} = \mathbf{LU}$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}\right)$$

Inverse matrix (7)

```
  if(j != imax){
       for (k = 0; k < nN; k++) {
         dum = a[imax][k];
         a[imax][k] = a[j][k];
          a[j][k] = dum;
       }
       *d = -(*d);                vv[imax] = vv[j];
  }

  indx[j] = imax;
  if(a[j][j] == 0.0) a[j][j] = TinyValue;
  if(j != nN - 1) {
       dum = 1.0 / (a[j][j]);
       for (i = j + 1; i < nN; i++)  a[i][j] *= dum;
  }
 }
 delete[] vv;
 return true;
}
```

$$\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}\right)$$

InverseMatrix: KhuGleBase.h/cpp
Forward/back substitution

LU 후진 대입

Inverse matrix (8)

```
void lubksb(double **a, int nN, int *indx, double *b) {
  int i, ii = -1, ip, j;
  double sum;

  for(i = 0; i < nN; i++) {
    ip = indx[i];          sum = b[ip];      b[ip] = b[i];
    if(ii >= 0) {
         for(j = ii; j <= i - 1; j++) sum -= a[i][j] * b[j];
    }
    else if(sum) ii = i;
    b[i] = sum;
  }

  for(i = nN - 1; i >= 0; i--) {
    sum = b[i];
    for (j = i + 1; j < nN; j++)
         sum -= a[i][j] * b[j];
    b[i] = sum / a[i][i];
  }
}
```

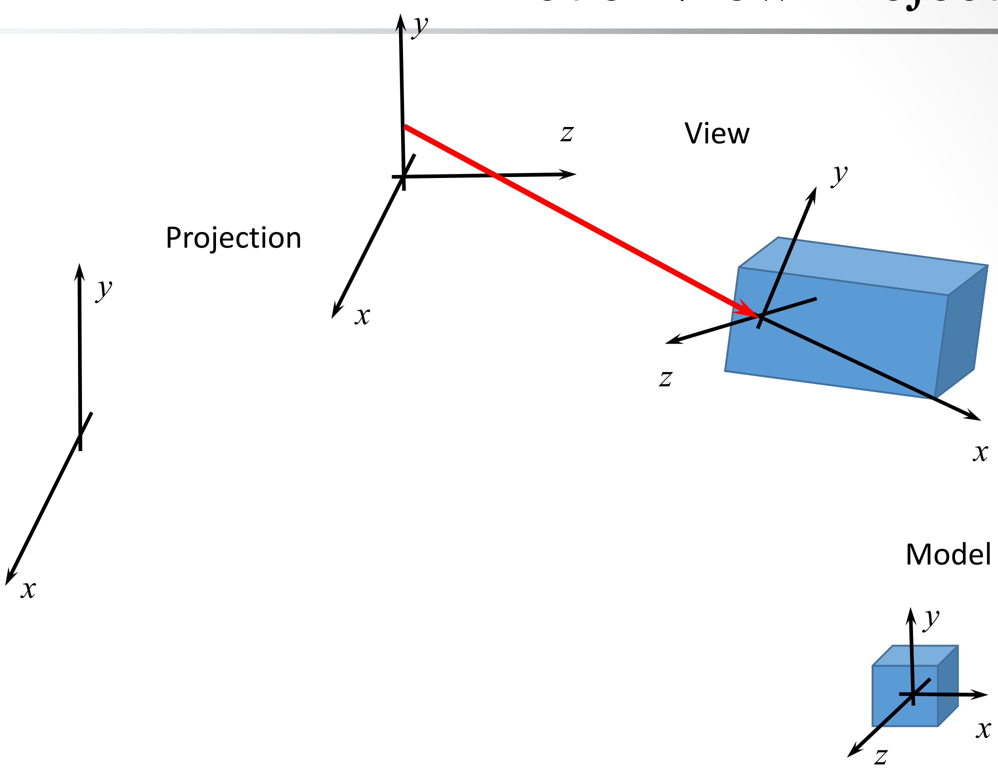
$$z_1 = 1$$
$$\alpha_{21} z_1 + z_2 = 0$$
$$\alpha_{31} z_1 + \alpha_{32} z_2 + z_3 = 0$$
$$\cdots$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ \alpha_{21} & 1 & 0 & 0 \\ \alpha_{31} & \alpha_{32} & 1 & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

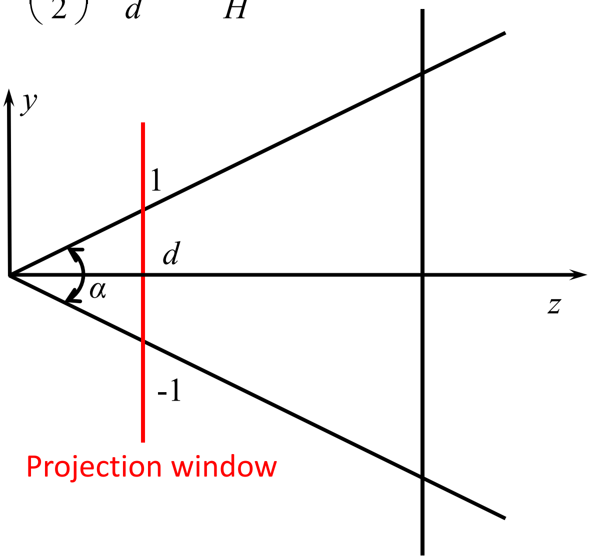
$$\beta_{44} b_4 = z_4$$
$$\beta_{33} b_3 + \beta_{34} b_4 = z_3$$
$$\beta_{22} b_2 + \beta_{23} b_3 + \beta_{24} b_4 = z_4$$
$$\cdots$$

Model/View/Projection



Projection (1)

$$\tan\left(\frac{\alpha}{2}\right)=\frac{1}{d},\ r=\frac{W}{H}$$



$$y_p : d = y : z$$

$$y_p=\frac{dy}{z}=\frac{y}{z\tan\left(\frac{\alpha}{2}\right)}$$

$$x_p : d = y : z$$

$$x_p=\frac{dx\left(H/W\right)}{z}=\frac{x}{rz\tan\left(\frac{\alpha}{2}\right)}$$

$$P=\begin{pmatrix} \frac{1}{r\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Projection (2)

- z range → [-1, 1]
- [near, far] → [-1, 1]

$$P=\begin{pmatrix} \frac{1}{r\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$z_p=\frac{Az+B}{z}=A+\frac{B}{z}\rightarrow$$

$$1=A+\frac{B}{far},\quad -1=A+\frac{B}{near}$$

$$2=\frac{B}{far}-\frac{B}{near}=\frac{B(near-far)}{far\cdot near}$$

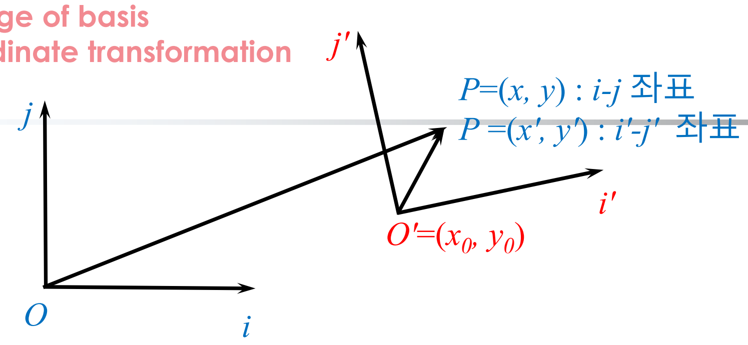
$$B=\frac{2\,far\times near}{near-far}$$

$$A=1\text{-}\frac{2\,far\times near}{near\text{ - }far}\frac{1}{far}=\frac{\text{-}near\text{ - }far}{near\text{ - }far}$$

$$P=\begin{pmatrix} \frac{1}{r\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{\text{-}near\text{ - }far}{near\text{ - }far} & \frac{2\,far\times near}{near-far} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Change of basis
Coordinate transformation

View (1)



$$\overrightarrow{OP} = x\vec{i} + y\vec{j}$$
$$\overrightarrow{O'P} = x'\vec{i}' + y'\vec{j}'$$
$$\overrightarrow{OO'} = x_0\vec{i} + y_0\vec{j}$$

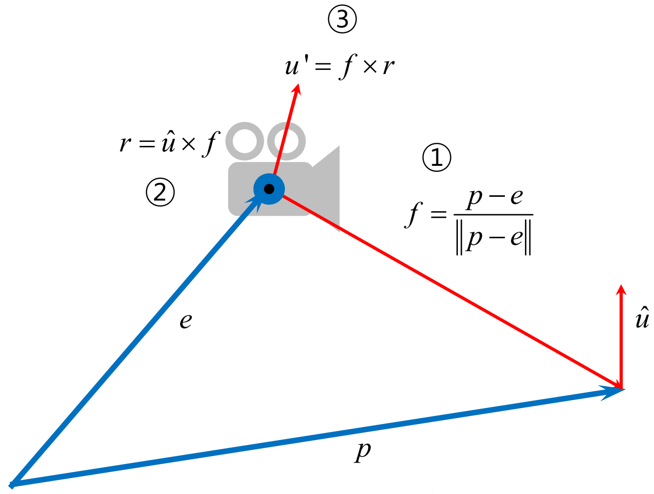
$$\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P}$$
$$x\vec{i} + y\vec{j} = x_0\vec{i} + y_0\vec{j} + x'\vec{i}' + y'\vec{j}'$$
$$(x - x_0)\vec{i} + (y - y_0)\vec{j} = x'\vec{i}' + y'\vec{j}'$$
$$\begin{pmatrix}\vec{i} & \vec{j}\end{pmatrix}\begin{pmatrix}x - x_0 \\ y - y_0\end{pmatrix} = \begin{pmatrix}\vec{i}' & \vec{j}'\end{pmatrix}\begin{pmatrix}x' \\ y'\end{pmatrix}$$
$$\begin{pmatrix}x - x_0 \\ y - y_0\end{pmatrix} = \begin{pmatrix}\vec{i} & \vec{j}\end{pmatrix}^{-1}\begin{pmatrix}\vec{i}' & \vec{j}'\end{pmatrix}\begin{pmatrix}x' \\ y'\end{pmatrix} = \begin{pmatrix}\vec{i}^T \\ \vec{j}^T\end{pmatrix}\begin{pmatrix}\vec{i}' & \vec{j}'\end{pmatrix}\begin{pmatrix}x' \\ y'\end{pmatrix}$$

$$\begin{pmatrix}x \\ y \\ z\end{pmatrix} = \begin{pmatrix}\vec{i}^T\vec{i}' & \vec{i}^T\vec{j}' & x_0 \\ \vec{j}^T\vec{i}' & \vec{j}^T\vec{j}' & y_0 \\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x' \\ y' \\ z'\end{pmatrix}$$

$$\begin{array}{cccc} & \vec{i} & \vec{j} & O' \end{array}$$
$$= \begin{pmatrix}\boxed{\begin{matrix} i'_x \\ i'_y \end{matrix}} & \boxed{\begin{matrix} j'_x \\ j'_y \end{matrix}} & \boxed{\begin{matrix} x_0 \\ y_0 \end{matrix}} \\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x' \\ y' \\ z'\end{pmatrix}$$

Change of basis
Coordinate transformation

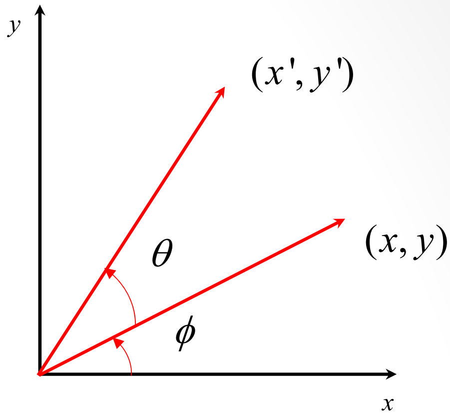
View (2)



$$\begin{pmatrix}x \\ y \\ z \\ w\end{pmatrix} = \begin{pmatrix}r_x & u'_x & f_x & C_x = e_x \\ r_y & u'_y & f_y & C_y = e_y \\ r_z & u'_z & f_z & C_z = e_z \\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x' \\ y' \\ z' \\ w'\end{pmatrix} = V^{-1}\begin{pmatrix}x' \\ y' \\ z' \\ w'\end{pmatrix}$$

Model (1)

$$x = r\cos\phi$$
$$y = r\sin\phi$$



$$\begin{aligned} x' &= r\cos(\phi+\theta) \\ &= r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ &= x\cos\theta - y\sin\theta \\ y' &= r\sin(\phi+\theta) \\ &= r\cos\phi\sin\theta + r\sin\phi\cos\theta \\ &= x\sin\theta + y\cos\theta \end{aligned}$$

Model (2)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \mathrm{y}\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \\ 1 \end{pmatrix}$$

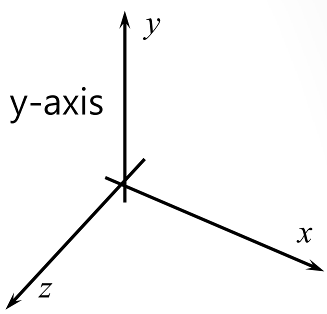
around x-axis

$R_x(\theta)$

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta + z\sin\theta \\ y \\ -x\sin\theta + z\cos\theta \\ 1 \end{pmatrix}$$

around y-axis

$R_y(\theta)$



$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ \mathrm{x}\sin\theta + y\cos\theta \\ z \\ 1 \end{pmatrix}$$

around z-axis

$R_z(\theta)$

Model (3)

$$R_z(\gamma)R_y(\beta)R_x(\alpha)$$

$$\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\gamma\cos\beta & -\sin\gamma & \cos\gamma\sin\beta & 0 \\ \sin\gamma\cos\beta & \cos\gamma & \sin\gamma\sin\beta & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha & \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha & 0 \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha & -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha & 0 \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Model (4)

- Translation

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{pmatrix}$$

- Scaling

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ 1 \end{pmatrix}$$

Main.cpp (1)

```
#include "KhuGleWin.h"
#include <iostream>

struct CKgTriangle{
  CKgVector3D v0, v1, v2;

  CKgTriangle()
    : v0(CKgVector3D()), v1(CKgVector3D()), v2(CKgVector3D()) {};
  CKgTriangle(CKgVector3D vv0, CKgVector3D vv1, CKgVector3D vv2)
    : v0(vv0), v1(vv1), v2(vv2) {};
};
```


Main.cpp (2)

```
class CKhuGle3DSprite : public CKhuGleSprite {
public:
  std::vector<CKgTriangle> SurfaceMesh;
  double **m_ProjectionMatrix;
  CKgVector3D m_CameraPos;

  CKhuGle3DSprite(int nW, int nH, double Fov,
    double Far, double Near, KgColor24 fgColor);
  ~CKhuGle3DSprite();
```

Main.cpp (3)

```
  static void DrawTriangle(unsigned char **R,
    unsigned char **G, unsigned char **B, int nW, int nH,
    int x0, int y0, int x1, int y1, int x2, int y2,
    KgColor24 Color24);

  static void MatrixVector44(CKgVector3D &out,
    CKgVector3D v, double **M);

  static double **ComputeViewMatrix(CKgVector3D Camera,
    CKgVector3D Target, CKgVector3D CameraUp);

  void Render();
  void MoveBy(double OffsetX, double OffsetY, double OffsetZ);
};
```


Main.cpp (4)

```
CKhuGle3DSprite::CKhuGle3DSprite(int nW, int nH, double Fov, double Far,
  double Near, KgColor24 fgColor) {
  m_fgColor = fgColor;
  m_CameraPos = CKgVector3D(0., -0.2, -2);

  m_ProjectionMatrix = dmatrix(4, 4);
  for(int r = 0 ; r < 4 ; ++r)
    for(int c = 0 ; c < 4 ; ++c)
      m_ProjectionMatrix[r][c] = 0.;

  m_ProjectionMatrix[0][0] = (double)nH/(double)nW * 1./tan(Fov/2.);
  m_ProjectionMatrix[1][1] = 1./tan(Fov/2.);
  m_ProjectionMatrix[2][2] = (-Near-Far) / (Near-Far);
  m_ProjectionMatrix[2][3] = 2.*(Far * Near) / (Near-Far);
  m_ProjectionMatrix[3][2] = 1.;
  m_ProjectionMatrix[3][3] = 0.;
```

$$P = \begin{pmatrix} \frac{1}{r\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{-near - far}{near - far} & \frac{2\,far \times near}{near - far} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Main.cpp (5)

```
 SurfaceMesh.push_back(CKgTriangle(CKgVector3D(-0.5, 0., -sqrt(3.)/6),
    CKgVector3D(0.5, 0., -sqrt(3.)/6), CKgVector3D(0., 0., sqrt(3.)/3)));
  SurfaceMesh.push_back(CKgTriangle(CKgVector3D(0., 0., sqrt(3.)/3),
    CKgVector3D(0.5, 0., -sqrt(3.)/6), CKgVector3D(0., sqrt(3.)/3, 0.)));
  SurfaceMesh.push_back(CKgTriangle(CKgVector3D(-0.5, 0., -sqrt(3.)/6),
    CKgVector3D(0, 0., sqrt(3.)/3), CKgVector3D(0., sqrt(3.)/3, 0.)));
  SurfaceMesh.push_back(CKgTriangle(CKgVector3D(0.5, 0., -sqrt(3.)/6),
    CKgVector3D(-0.5, 0., -sqrt(3.)/6), CKgVector3D(0., sqrt(3.)/3, 0.)));
};

CKhuGle3DSprite::~CKhuGle3DSprite() {
  free_dmatrix(m_ProjectionMatrix, 4, 4);
};
```


Main.cpp (6)

```
void CKhuGle3DSprite::DrawTriangle(unsigned char **R,
  unsigned char **G, unsigned char **B,
  int nW, int nH, int x0, int y0, int x1, int y1,
  int x2, int y2, KgColor24 Color24) {

  CKhuGleSprite::DrawLine(R, G, B, nW, nH, x0, y0, x1, y1, Color24);
  CKhuGleSprite::DrawLine(R, G, B, nW, nH, x1, y1, x2, y2, Color24);
  CKhuGleSprite::DrawLine(R, G, B, nW, nH, x2, y2, x0, y0, Color24);
}
```

Main.cpp (7)

```
void CKhuGle3DSprite::MatrixVector44(CKgVector3D &out,
  CKgVector3D v, double **M) {
  out.x = v.x*M[0][0] + v.y*M[0][1] + v.z*M[0][2] + M[0][3];
  out.y = v.x*M[1][0] + v.y*M[1][1] + v.z*M[1][2] + M[1][3];
  out.z = v.x*M[2][0] + v.y*M[2][1] + v.z*M[2][2] + M[2][3];

  double w = v.x*M[3][0] + v.y*M[3][1] + v.z*M[3][2]
        + M[3][3];

  if(fabs(w) > 0)
    out = (1./w)* out;
}
```


Main.cpp (8)

```
double **CKhuGle3DSprite::ComputeViewMatrix(CKgVector3D Camera,
        CKgVector3D Target, CKgVector3D CameraUp) {

  CKgVector3D Forward = Target-Camera;

  Forward.Normalize();
  CameraUp.Normalize();

  CKgVector3D Right = CameraUp.Cross(Forward);
  CKgVector3D Up = Forward.Cross(Right);

  double **RT = dmatrix(4, 4);
  double **View = dmatrix(4, 4);
```

Main.cpp (9)

```
  RT[0][0] = Right.x;     RT[1][0] = Right.y;
  RT[2][0] = Right.z;     RT[3][0] = 0.;

  RT[0][1] = Up.x;        RT[1][1] = Up.y;
  RT[2][1] = Up.z;        RT[3][1] = 0.;

  RT[0][2] = Forward.x;   RT[1][2] = Forward.y;
  RT[2][2] = Forward.z;   RT[3][2] = 0.;

  RT[0][3] = Camera.x;    RT[1][3] = Camera.y;
  RT[2][3] = Camera.z;    RT[3][3] = 1.;

  bool bInverse = InverseMatrix(RT, View, 4);
  free_dmatrix(RT, 4, 4);

  if(bInverse) return View;
  return nullptr;
}
```

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} r_x & u'_x & f_x & C_x \\ r_y & u'_y & f_y & C_y \\ r_z & u'_z & f_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = V^{-1} \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

Main.cpp (10)

```
void CKhuGle3DSprite::Render() {
  if(!m_Parent) return;
  double NewX = m_CameraPos.x*cos(Pi/1000.) - m_CameraPos.z*sin(Pi/1000.);
  double NewZ = m_CameraPos.x*sin(Pi/1000.) + m_CameraPos.z*cos(Pi/1000.);
  m_CameraPos.x = NewX;
  m_CameraPos.z = NewZ;

  CKhuGleLayer *Parent = (CKhuGleLayer *)m_Parent;
  double **ViewMatrix = ComputeViewMatrix(m_CameraPos,
         CKgVector3D(0., 0., 0.), CKgVector3D(0., 1., 0.));
  if(ViewMatrix == nullptr) return;

  for(auto &Triangle: SurfaceMesh) {
    CKgVector3D Side01, Side02, Normal;
    Side01 = Triangle.v1 - Triangle.v0;
    Side02 = Triangle.v2 - Triangle.v0;
    Normal = Side01.Cross(Side02);
    Normal.Normalize();
```

Main.cpp (11)

```
    CKgTriangle ViewTriangle;
    CKgTriangle Projected;
    if(Normal.Dot(Triangle.v0 - m_CameraPos) < 0.)
    {
      MatrixVector44(ViewTriangle.v0, Triangle.v0, ViewMatrix);
      MatrixVector44(ViewTriangle.v1, Triangle.v1, ViewMatrix);
      MatrixVector44(ViewTriangle.v2, Triangle.v2, ViewMatrix);

      MatrixVector44(Projected.v0, ViewTriangle.v0, m_ProjectionMatrix);
      MatrixVector44(Projected.v1, ViewTriangle.v1, m_ProjectionMatrix);
      MatrixVector44(Projected.v2, ViewTriangle.v2, m_ProjectionMatrix);

      Projected.v0.x += 1.;              Projected.v0.y += 1.;
      Projected.v1.x += 1.;              Projected.v1.y += 1.;
      Projected.v2.x += 1.;              Projected.v2.y += 1.;
```


Main.cpp (12)

```
      Projected.v0.x *= Parent->m_nW/2.;
      Projected.v0.y *= Parent->m_nH/2.;
      Projected.v1.x *= Parent->m_nW/2.;
      Projected.v1.y *= Parent->m_nH/2.;
      Projected.v2.x *= Parent->m_nW/2.;
      Projected.v2.y *= Parent->m_nH/2.;
      Projected.v0.x -= 1.;              Projected.v0.y -= 1.;
      Projected.v1.x -= 1.;              Projected.v1.y -= 1.;
      Projected.v2.x -= 1.;              Projected.v2.y -= 1.;

      DrawTriangle(Parent->m_ImageR, Parent->m_ImageG, Parent->m_ImageB,
        Parent->m_nW, Parent->m_nH,
        (int)Projected.v0.x, (int)Projected.v0.y,
        (int)Projected.v1.x, (int)Projected.v1.y,
        (int)Projected.v2.x, (int)Projected.v2.y, m_fgColor);
    }
  }

  free_dmatrix(ViewMatrix, 4, 4);
}
```

Main.cpp (13)

```
void CKhuGle3DSprite::MoveBy(double OffsetX, double OffsetY,
  double OffsetZ) {
  for(auto &Triangle: SurfaceMesh) {
    Triangle.v0 = Triangle.v0 + CKgVector3D(OffsetX, OffsetY, OffsetZ);
    Triangle.v1 = Triangle.v1 + CKgVector3D(OffsetX, OffsetY, OffsetZ);
    Triangle.v2 = Triangle.v2 + CKgVector3D(OffsetX, OffsetY, OffsetZ);
  }
}

class CThreeDim : public CKhuGleWin {
public:
  CKhuGleLayer *m_pGameLayer;

  CKhuGle3DSprite *m_pObject3D;

  CThreeDim(int nW, int nH);
  void Update();

  CKgPoint m_LButtonStart, m_LButtonEnd;
  int m_nLButtonStatus;
};
```


Main.cpp (14)

```
CThreeDim::CThreeDim(int nW, int nH) : CKhuGleWin(nW, nH) {
  m_nLButtonStatus = 0;

  m_Gravity = CKgVector2D(0., 98.);
  m_AirResistance = CKgVector2D(0.1, 0.1);

  m_pScene = new CKhuGleScene(640, 480,
    KG_COLOR_24_RGB(100, 100, 150));

  m_pGameLayer = new CKhuGleLayer(600, 420,
    KG_COLOR_24_RGB(150, 150, 200), CKgPoint(20, 30));
  m_pScene->AddChild(m_pGameLayer);

  m_pObject3D = new CKhuGle3DSprite(m_pGameLayer->m_nW,
    m_pGameLayer->m_nH, Pi/2., 1000., 0.1,
    KG_COLOR_24_RGB(255, 0, 255));

  m_pGameLayer->AddChild(m_pObject3D);
}
```

Main.cpp (15)

```
void CThreeDim::Update() {
  if(m_bKeyPressed[VK_DOWN])
    m_pObject3D->MoveBy(0, 0.0005, 0);

  m_pScene->Render();
  DrawSceneTextPos("3D Rendering", CKgPoint(0, 0));

  CKhuGleWin::Update();
}

int main() {
  CThreeDim *pThreeDim = new CThreeDim(640, 480);

  KhuGleWinInit(pThreeDim);

  return 0;
}
```


Practice II

- Model matrix

Advanced Courses

- Depth
- Texture