3. Преобразования координат

Вектор в однородной форме

$$\mathbf{A} = \begin{vmatrix} x \\ y \\ 0 \end{vmatrix}$$

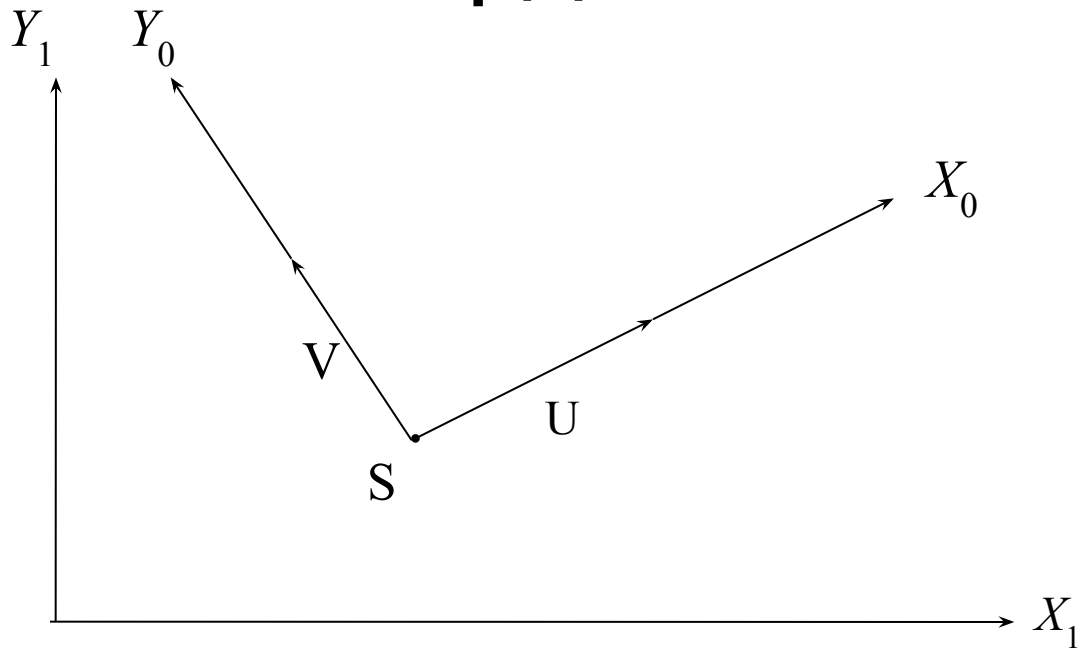
$$\begin{vmatrix} x_1 \\ y_1 \\ 0 \end{vmatrix} + \begin{vmatrix} x_2 \\ y_2 \\ 0 \end{vmatrix} = \begin{vmatrix} x_1 + x_2 \\ y_1 + y_2 \\ 0 \end{vmatrix}$$

$$\begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} + \begin{vmatrix} x_2 \\ y_2 \\ 0 \end{vmatrix} = \begin{vmatrix} x_1 + x_2 \\ y_1 + y_2 \\ 1 \end{vmatrix}$$

$$\begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} - \begin{vmatrix} x_2 \\ y_2 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1 - x_2 \\ y_1 - y_2 \\ 0 \end{vmatrix}$$

$$m \begin{vmatrix} x_1 \\ y_1 \\ 0 \end{vmatrix} = \begin{vmatrix} mx_1 \\ my_1 \\ 0 \end{vmatrix}$$

Переход к новой системе координат

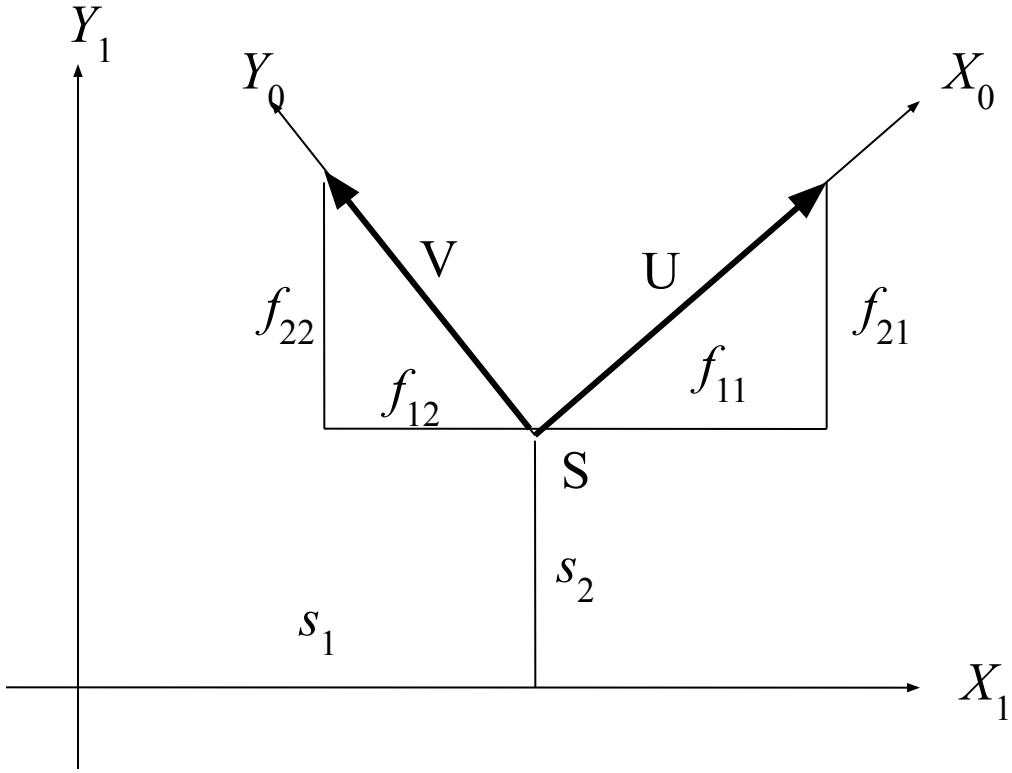


$$A_1 = F_{01} A_0$$

$$A_0 = F_{10} A_1$$

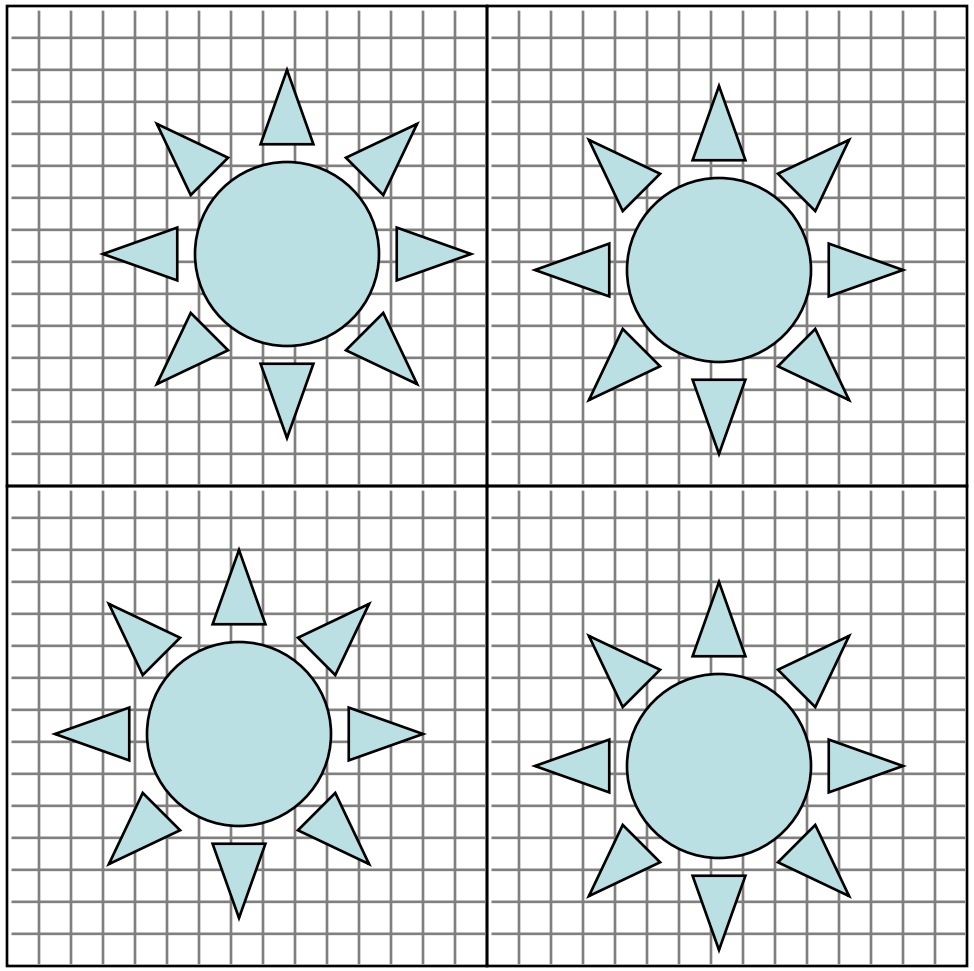
$$F_{10} = F_{01}^{-1}$$

Координатный фрейм

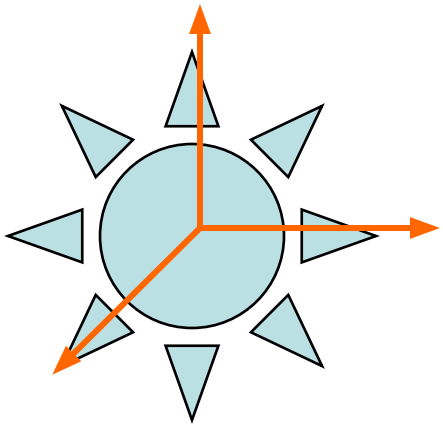


$$F_{01} = \begin{vmatrix} f_{11} & f_{12} & s_1 \\ f_{21} & f_{22} & s_2 \\ 0 & 0 & 1 \end{vmatrix}$$

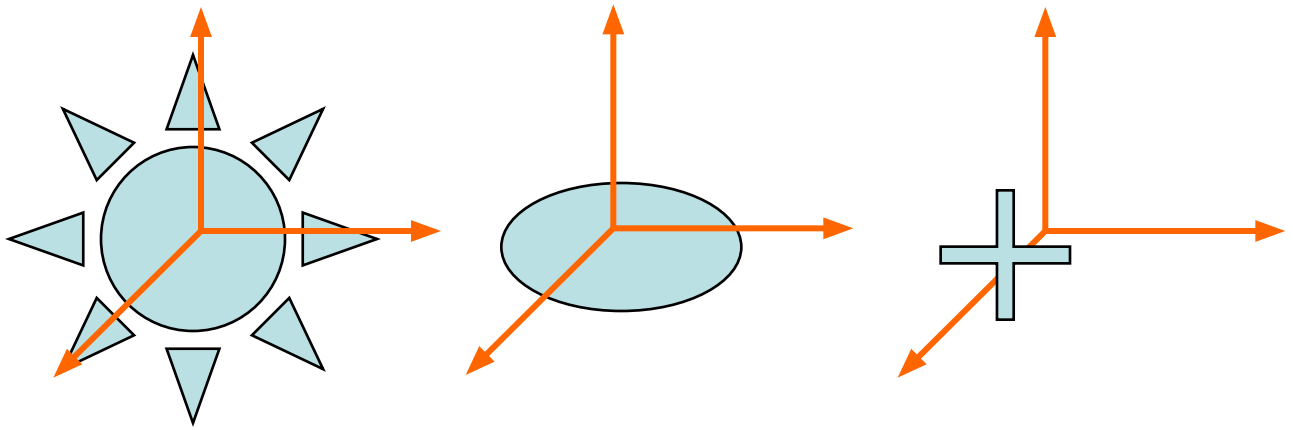
Объектная система координат



При создании (моделировании) объекта используется т. н. ***объектная*** (или ***локальная***) система координат.

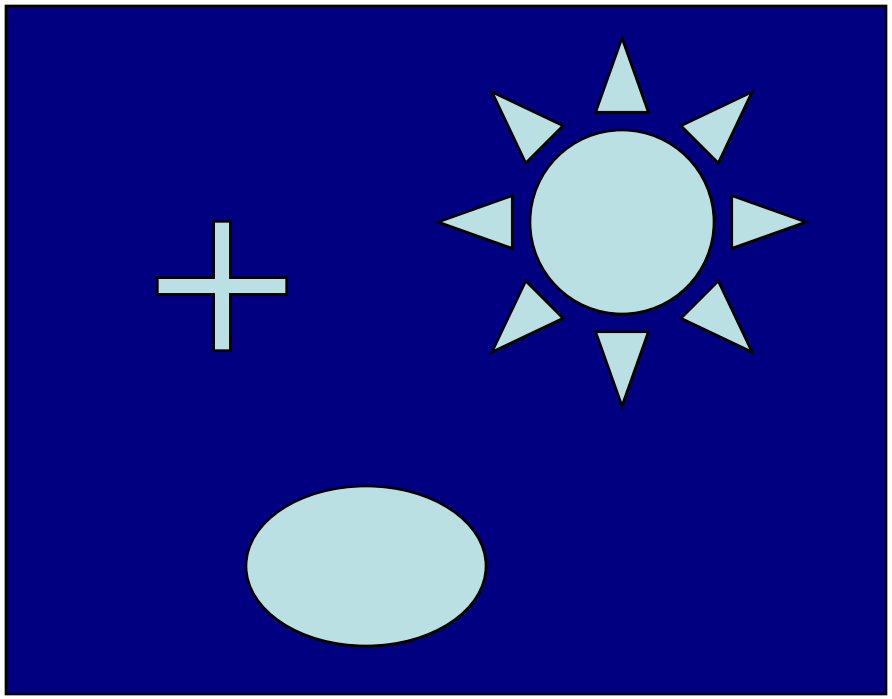


Объектная система координат



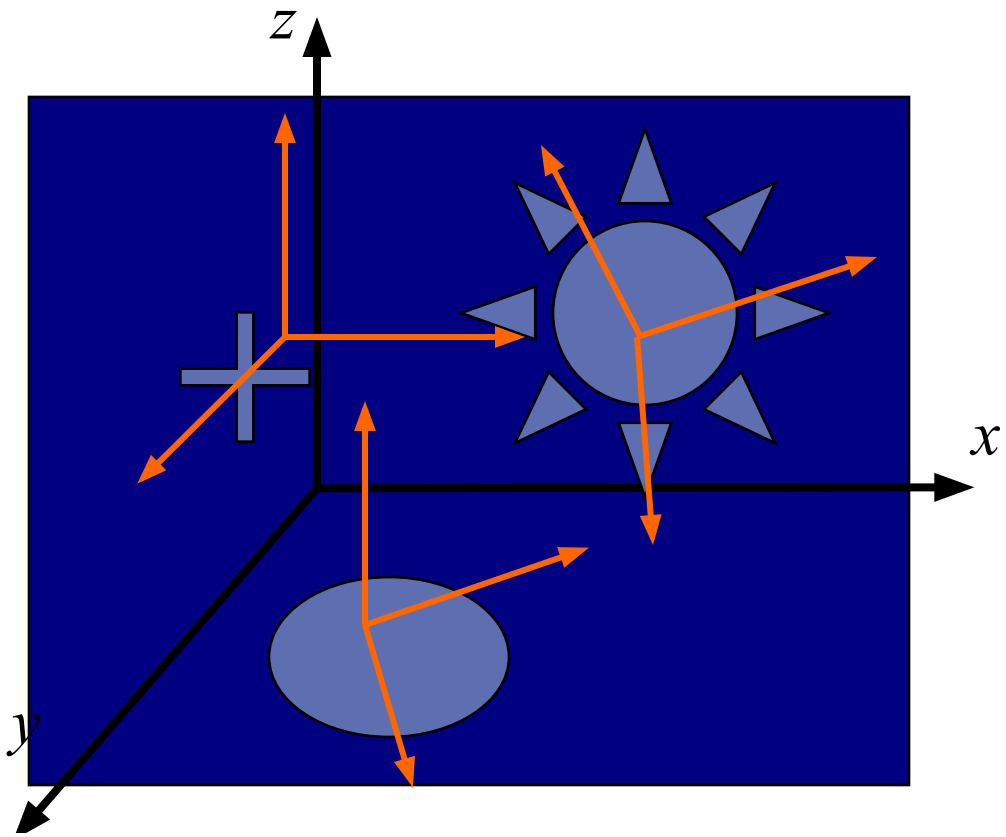
Использование объектных координат сильно облегчает жизнь при создании моделей объектов. Разные модели никак не зависят друг от друга - можно использовать разные масштабы, разные точки отсчета и т. п.

Мировая система координат



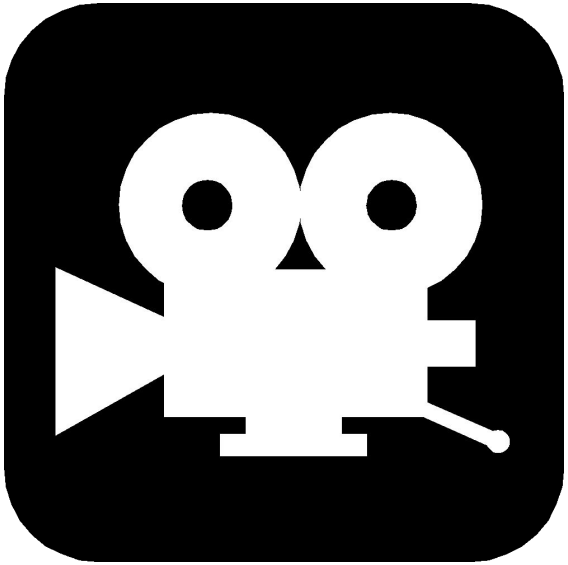
Для того, что бы составить из нескольких объектов сцену, необходимо расположить и ориентировать объекты друг относительно друга определенным образом.

Мировая система координат



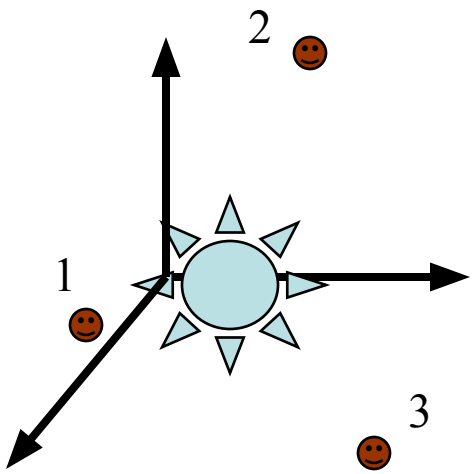
М***ировой*** называют систему координат, используемую как единую систему отсчета для **всей** сцены.

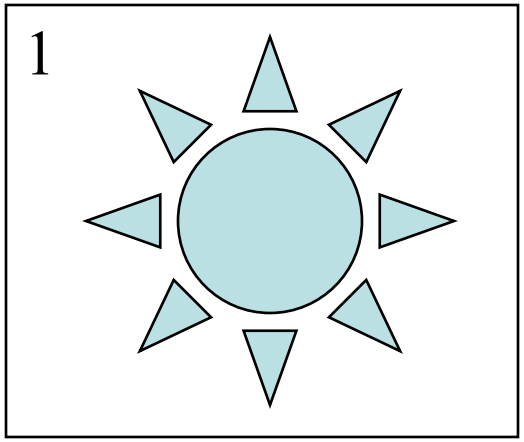
Система координат камеры

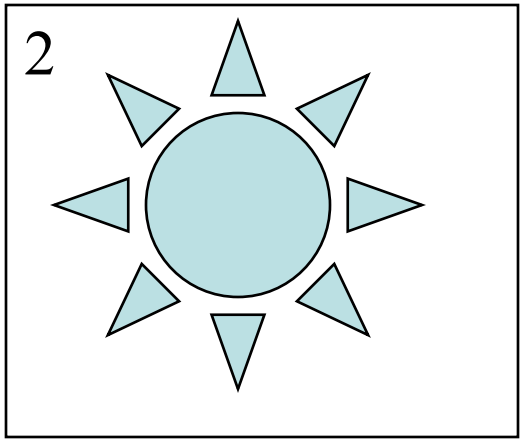


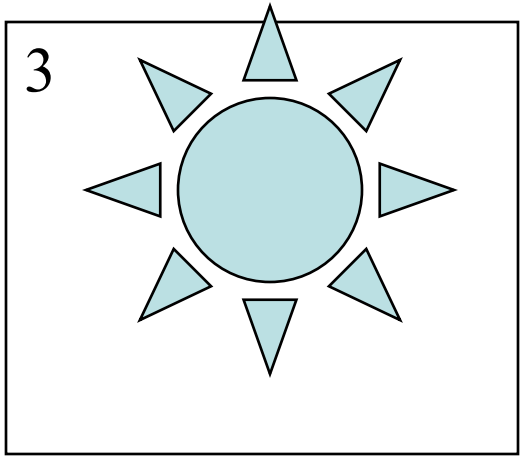
После того, как объекты помещены на сцену (в мировую систему координат!), самое время подумать о том, где и как установить камеру.

Система координат камеры









Помещая камеру в различные точки сцены мы имеем возможность акцентировать внимание зрителя на различных частях сцены

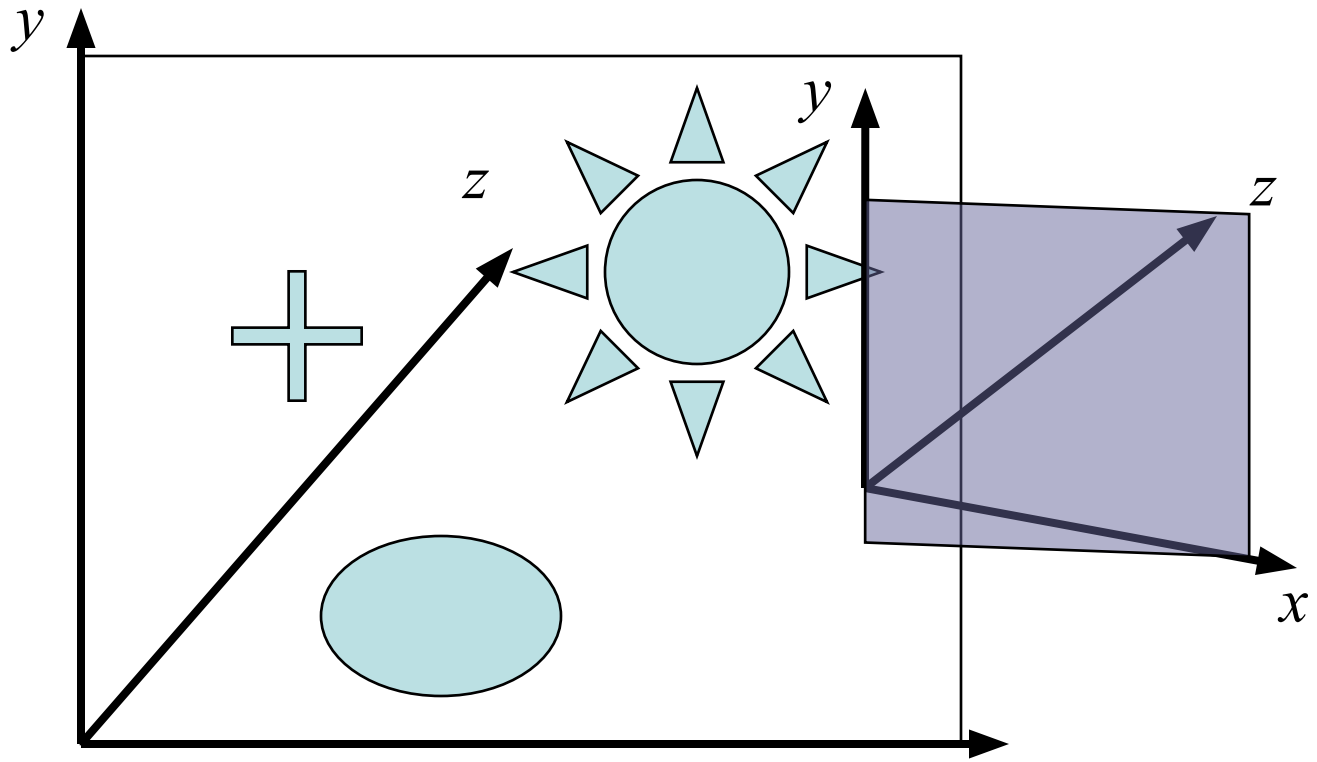
Система координат камеры

«П…вую
си…ны.

Один и тот же объект в разных «камерных» системах координат

Система координат, в которой точка отсчета привязана к камере, используется в процессе отображения объектов на экран.

Система координат камеры

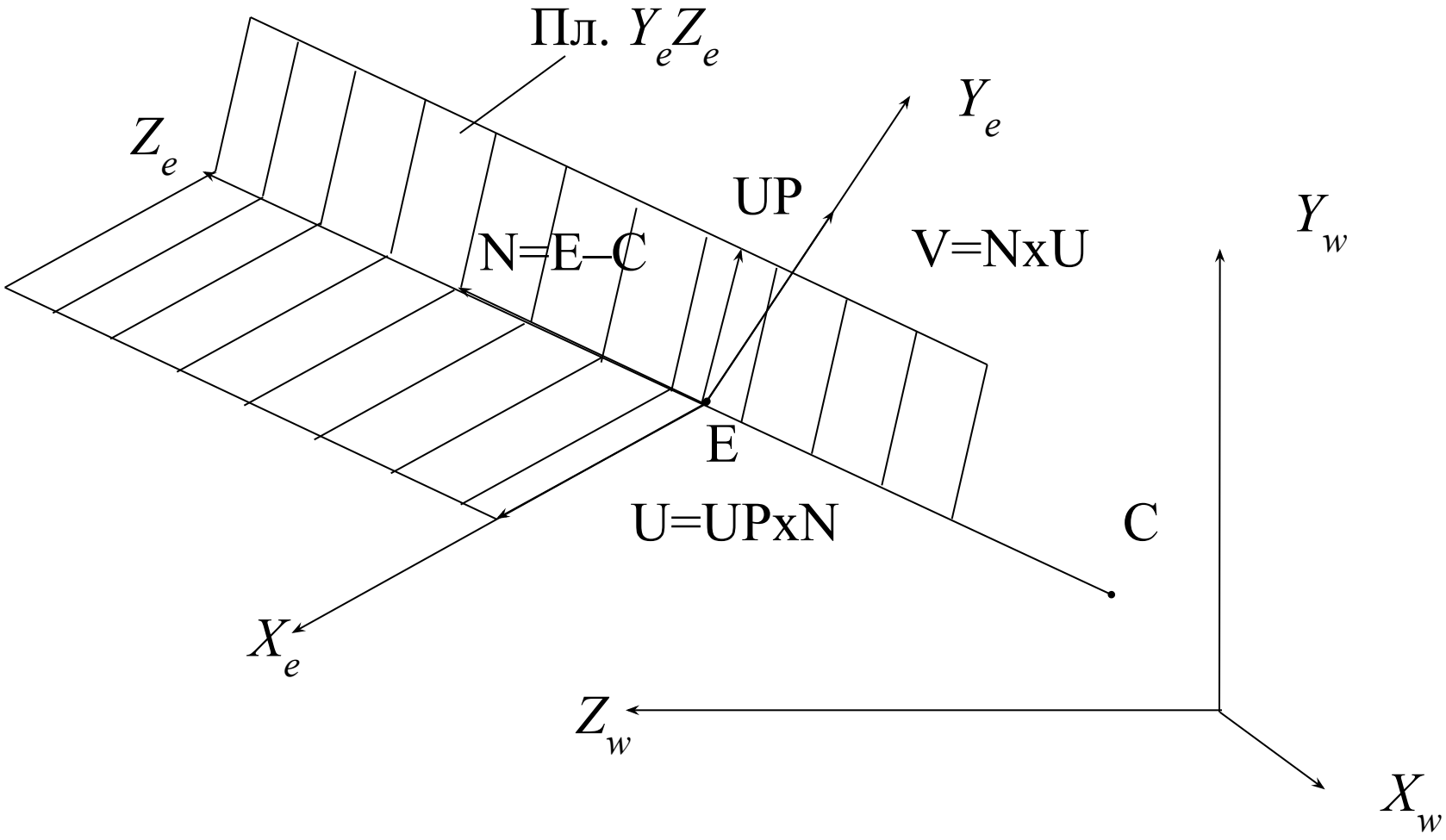


В библиотеках Direct3D и OpenGL оси системы координат камеры ориентированы следующим образом: *Ox* - слева на право, *Oy* - с низу вверх, *Oz* - вглубь экрана для D3D и наоборот для OpenGL.
Плоскость *Oxy* совпадает с плоскостью экрана.

Положение наблюдателя

```
void gluLookAt(
   GLdouble ex,   GLdouble ey,  GLdouble ez,
   GLdouble cx,   GLdouble cy,   GLdouble cz,
   GLdouble upx, GLdouble upy, GLdouble upz
);
```

Система координат пользователя



Определение системы координат наблюдателя

$$\mathbf{E} = \begin{bmatrix} ex \\ ey \\ ez \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} cx \\ cy \\ cz \\ 1 \end{bmatrix}, \mathbf{UP} = \begin{bmatrix} upx \\ upy \\ upz \\ 1 \end{bmatrix}$$

$$\frac{\mathbf{N}}{|\mathbf{N}|} = \frac{\mathbf{E}-\mathbf{C}}{|\mathbf{E}-\mathbf{C}|} = \begin{bmatrix} nx \\ ny \\ nz \\ 0 \end{bmatrix}, \frac{\mathbf{U}}{|\mathbf{U}|} = \frac{\mathbf{UP}\times\mathbf{N}}{|\mathbf{UP}\times\mathbf{N}|} = \begin{bmatrix} ux \\ uy \\ uz \\ 0 \end{bmatrix}, \frac{\mathbf{V}}{|\mathbf{V}|} = \frac{\mathbf{N}\times\mathbf{U}}{|\mathbf{N}\times\mathbf{U}|} = \begin{bmatrix} vx \\ vy \\ vz \\ 0 \end{bmatrix}$$

$$\mathbf{F}_{ew} = \begin{bmatrix} ux & vx & nx & ex \\ uy & vy & ny & ey \\ uz & vz & nz & ez \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{F}_{we} = \mathbf{F}_{ew}^{-1}$$

Пример

$$\mathbf{E}=\begin{bmatrix}4\\4\\4\\1\end{bmatrix},\mathbf{C}=\begin{bmatrix}0\\1\\0\\1\end{bmatrix},\mathbf{UP}=\begin{bmatrix}0\\1\\0\\1\end{bmatrix}$$

$$\frac{\mathbf{N}}{|\mathbf{N}|}=\frac{\mathbf{E}-\mathbf{C}}{|\mathbf{E}-\mathbf{C}|}=\begin{bmatrix}0.625\\0.469\\0.625\\0\end{bmatrix},\frac{\mathbf{U}}{|\mathbf{U}|}=\frac{\mathbf{UP}\times\mathbf{N}}{|\mathbf{UP}\times\mathbf{N}|}=\begin{bmatrix}0.707\\0\\-0.707\\0\end{bmatrix},\frac{\mathbf{V}}{|\mathbf{V}|}=\frac{\mathbf{N}\times\mathbf{U}}{|\mathbf{N}\times\mathbf{U}|}=\begin{bmatrix}-0.331\\0.881\\-0.331\\0\end{bmatrix}$$

$$\mathbf{F}_{ew}=\begin{bmatrix}0.707&-0.331&0.625&4\\0&0.881&0.469&4\\-0.707&-0.331&0.625&4\\0&0&0&1\end{bmatrix},\mathbf{F}_{we}=\mathbf{F}_{ew}^{-1}=\begin{bmatrix}0.707&0&-0.707&0\\-0.331&0.881&-0.331&-0.884\\0.625&0.469&0.625&-6.876\\0&0&0&1\end{bmatrix}$$

Пример управления камерой

```
#include <GL/glut.h>
#define _USE_MATH_DEFINES
#include <math.h>

struct Point3
{
	double x;
	double y;
	double z;
};

struct Vector3
{
	double x;
	double y;
	double z;
};

Point3 eye, look;
Vector3 up;
Vector3 u, v, n;
```

Пример управления камерой

```
//Загрузка матрицы моделирования-вида существующими для камеры величинами
void setModelViewMatrix(void)
{
    float m[16 ];
    Vector3 eVec;

    eVec.x = eye.x; eVec.y = eye.y; eVec.z = eye.z;

    m[0] = u.x; m[4] = u.y; m[8] = u.z;  m[12] = -eVec.x * u.x - eVec.y * u.y - eVec.z * u.z;
    m[1] = v.x; m[5] = v.y; m[9] = v.z;  m[13] = -eVec.x * v.x - eVec.y * v.y - eVec.z * v.z;
    m[2] = n.x; m[6] = n.y; m[10] = n.z; m[14] = -eVec.x * n.x - eVec.y * n.y - eVec.z * n.z;
    m[3] = 0.0; m[7] = 0.0; m[11] = 0.0; m[15] = 1.0;

    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(m);
}
```

Пример управления камерой

```
// Создание матрицы моделирования-вида
void set(Point3 Eye, Point3 Look, Vector3 Up)
{
    eye = Eye; look = Look; up = Up;
    n.x = eye.x - look.x; n.y = eye.y - look.y; n.z = eye.z - look.z;
    u.x = up.y * n.z - up.z * n.y; u.y = up.z * n.x - up.x * n.z; u.z = up.x * n.y - up.y * n.x;

    double norm;

    norm = sqrt(n.x * n.x + n.y * n.y + n.z * n.z);
    n.x = n.x / norm; n.y = n.y / norm; n.z = n.z / norm;
    norm = sqrt(u.x * u.x + u.y * u.y + u.z * u.z);
    u.x = u.x / norm; u.y = u.y / norm; u.z = u.z / norm;

    v.x = n.y * u.z - n.z * u.y; v.y = n.z * u.x - n.x * u.z; v.z = n.x * u.y - n.y * u.x;

    setModelViewMatrix();
}
```

Пример управления камерой

```
//Скольжение
void slide(float delU, float delV, float delN)
{
    eye.x += delU * u.x + delV * v.x + delN * n.x;
    eye.y += delU * u.y + delV * v.y + delN * n.y;
    eye.z += delU * u.z + delV * v.z + delN * n.z;
    setModelViewMatrix();
}

//Крен
void roll(float angle)
{
    float cs = cos(M_PI / 180 * angle);
    float sn = sin(M_PI / 180 * angle);
    Vector3 t = u;
    u.x = cs * t.x - sn * v.x; u.y = cs * t.y - sn * v.y; u.z = cs * t.z - sn * v.z;
    v.x = sn * t.x + cs * v.x; v.y = sn * t.y + cs * v.y; v.z = sn * t.z + cs * v.z;
    setModelViewMatrix();
}
```

Пример управления камерой

```
//Тангаж
void pitch(float angle)
{
    float cs = cos(M_PI / 180 * angle);
    float sn = sin(M_PI / 180 * angle);
    Vector3 t = v;
    v.x = cs * t.x - sn * n.x; v.y = cs * t.y - sn * n.y; v.z = cs * t.z - sn * n.z;
    n.x = sn * t.x + cs * n.x; n.y = sn * t.y + cs * n.y; n.z = sn * t.z + cs * n.z;
    setModelViewMatrix();
}
//Рыскание
void yaw(float angle)
{
    float cs = cos(M_PI / 180 * angle);
    float sn = sin(M_PI / 180 * angle);
    Vector3 t = u;
    u.x =  cs * t.x + sn * n.x; u.y =  cs * t.y + sn * n.y; u.z =  cs * t.z + sn * n.z;
    n.x = -sn * t.x + cs * n.x; n.y = -sn * t.y + cs * n.y; n.z = -sn * t.z + cs * n.z;
    setModelViewMatrix();
}
```

Пример управления камерой

```
//Поворот камеры вокруг оси v
void rotate(float angle)
{
    float cs = cos(M_PI / 180 * angle);
    float sn = sin(M_PI / 180 * angle);
    Point3 Eye, Look;
    Vector3 Up;

    Eye.x = cs * eye.x - sn * eye.z;
    Eye.y = eye.y;
    Eye.z = sn * eye.x + cs * eye.z;
    Look = look;
    Up = up;
    set(Eye, Look, Up);
}
void setShape(float left, float right, float bottom, float top, float near, float far)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(left, right, bottom, top, near, far);
}
```

Пример управления камерой

```
void myKeyboard(unsigned char key, int x, int y)
{
    switch(key)
    {
      case '1': slide(0, 0, 0.2); break;   // скольжение вперед
      case '2': slide(0, 0, -0.2); break;  // скольжение назад
      case '3': pitch(-1.0); break;        // тангаж вверх
      case '4': pitch(1.0); break;         // тангаж ввниз
      case '5': roll(-1.0); break;         // крен влево
      case '6': roll(1.0); break;          // крен вправо
      case '7': yaw(-1.0); break;          // рыскание влево
      case '8': yaw(1.0); break;    // рыскание вправо
      case '9': rotate(1.0); break;
      case '0': rotate(-1.0); break;
    }
    glutPostRedisplay();
}
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutWireTeapot(1.0);
    glFlush();
    glutSwapBuffers();
}
```

Пример управления камерой

```
void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Пилотирование камеры вокруг чайника");
    glClearColor(1.0f,1.0f,1.0f,1.0f);
    glColor3f(0.0f,0.0f,0.0f);
    glViewport(0, 0, 640, 480);
    Point3  Eye = {4, 4, 4}, look = {0, 0, 0};
    Vector3 up = {0, 1, 0};
    set(Eye, look, up);
    setShape(-2.0f, 2.0f, -2.0f, 2.0f, 0.0f, 10.0f);
    glutDisplayFunc(myDisplay);
    glutKeyboardFunc(myKeyboard);
    glutMainLoop();
}
```

Пример управления камерой