Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет Cистемы управления и робототехники

A diagram of a cube

Description automatically generatedA diagram of a cube

Description automatically generatedA diagram of a cube

Description automatically generatedA diagram of a cube

Description automatically generated

**Отчет по лабораторной работе**

**№3 « Матрицы в 3D-графике»**

**Преподаватель**:

Перегудин А. А.,

Ассистент фак. СУиР

**Выполнила**:

студентка гр. R3235

Нгуен Кхань Нгок

A colorful cube with a square in the center

Description automatically generatedA graph showing a cube and a cube

Description automatically generated with medium confidenceA diagram of a cube

Description automatically generated

Санкт-Петербург 2023

Результаты

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  | |

Первое изображение использует другую формулу перспективы, чем второе изображение 2.

Original Cube

figure()

verticesCube = [

-1, 1, 1,-1,-1, 1, 1 ,-1;

-1,-1, 1, 1,-1,-1, 1, 1;

-1,-1,-1,-1, 1, 1, 1, 1;

1, 1, 1, 1, 1, 1, 1, 1

];

facesCube = [

1, 2, 6, 5;

2, 3, 7, 6;

3, 4, 8, 7;

4, 1, 5, 8;

1, 2, 3, 4;

5, 6, 7, 8

];

DrawShape(verticesCube, facesCube, 'flat')

title("Original Cube");

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

axis equal;

view(3)

hold on

function DrawShape (vertices , faces , flat )

patch ('Vertices', ( vertices (1:3,:) ./ vertices (4 ,:))', 'Faces', faces ,'FaceVertexCData',hsv(6),'FaceColor', flat, 'facealpha', 0.3)

end

**Задание 1**. Создайте кубик

# как работает данный код.

# A cube with different colored squares Description automatically generated

*1-ая пов-ость*

*2-ая пов-ость*

*3-ая пов-ость*

*4-ая пов-ость*

*5-ая пов-ость*

*6-ая пов-ость*

***Function DrawShape*** *:* Рисовать фигуру

- Через параметр «vetices» передайте матрицу вершин, рассчитанную путем деления x, y, z (строки 1, 2, 3) вершин на w (строка 4).

- Через параметр «Faces» передайте матрицу facesCube, чтобы указать вершины для объединения их в плоскость.- FaceColor: передача цвета

8

D’(-1, 1, -1)

5

A’(-1, -1, 1)

7

C’(1, 1, 1)

6

B’(-1, 1, -1)

D(-1, 1, -1)

4

A(-1, -1, -1)

1

C(1, 1, -1)

# 

3

2

B(-1, -1, -1)

**Почему мы используем четырехкомпонентный вектор, а не трех? Как задать другие фигуры?**

* В компьютерной графике и 3D-преобразованиях вместо трехкомпонентного вектора используется четырехкомпонентный вектор для представления точек в однородных координатах.
* Трехмерная точка с координатами {x, y, z} и точка с однородными координатами {x, y, z, w} эквивалентны условию w = 1. Трехмерную точку можно определить как точку с однородными координатами, если мы писать:

**P={x, y, z, w = 1}**

* Дополнительный компонент, часто обозначаемый как «w», обеспечивает большую гибкость преобразований и перспективного проецирования.
* Чтобы преобразовать эту точку обратно в 3D, нам нужно будет разделить координаты точек {x, y, z} на w. Несмотря на то, что четвертая строка матрицы преобразования 4x4 всегда равна 0, 0, 0, 1, это означает, что из-за способа перемножения точек и матриц четвертая координата преобразованной точки всегда равна 1.
* Чтобы настроить другие фигуры, вы должны определить вершины фигуры как четырехкомпонентный вектор, аналогично verticesCube. Каждый столбец представляет вершину, а строки соответствуют координатам x, y, z и w соответственно. Вы можете изменять координаты вершин для создания различных фигур в трехмерном пространстве.
* Трехмерная декартова точка P, преобразованная в точку с однородными координатами {x, y, z, w = 1} и умноженная на матрицу аффинного преобразования 4x4, всегда дает точку P' с однородными координатами и координата w которой всегда равна в 1. Таким образом, преобразование преобразованной точки P' с однородными координатами {x', y', z', w'} обратно в трехмерную декартову координату {x'/w', y'/w', z'/w '}, не требует явной нормализации однородных координат преобразованной точки на.
* Используя однородные координаты, мы можем представлять перемещения, вращения, масштабирование и перспективные проекции с помощью матричных операций, что упрощает математические вычисления в 3D-графике.

A cat holding a wrench

Description automatically generatedA cat holding a wrench

Description automatically generatedA colorful cube with a square in the center

Description automatically generatedA colorful cube with a square in the center

Description automatically generated

**Задание 2.** Измените масштаб кубика

figure('Name','Scale', NumberTitle='off')

verticesCube2 = getTranslated\_Scale(0,0,0,0.5, 0.5, 0.5)\*verticesCube;

DrawShape(verticesCube, facesCube,'flat');

DrawShape(verticesCube2, facesCube, 'black');

title("Scale");

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

axis equal;

view(3)

hold on

A diagram of a cube

Description automatically generatedA diagram of a cube

Description automatically generated

**Задание 3.** Переместите кубик

figure('Name','Translation', NumberTitle='off')

verticesCube3 = getTranslated\_Scale(1, 1, 2, 0.5, 0.5, 0.5)\*verticesCube;

DrawShape(verticesCube, facesCube, 'flat')

DrawShape(verticesCube3, facesCube, 'red')

title("Translation")

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

axis equal;

view(3)

hold on

A graph showing a cube and a cube

Description automatically generated with medium confidenceA graph showing a cube and a cube

Description automatically generated with medium confidence

**Задание 4.** Вращение кубика

figure('Name','Rotation', NumberTitle='off')

verticesCube4 = getTranslated\_Scale(0,0,3,1,1,1)\* getRotated\_x(30)\*verticesCube;

DrawShape(verticesCube, facesCube, 'flat');

DrawShape(verticesCube4, facesCube, 'red');

title("Rotation")

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

A diagram of a cube

Description automatically generatedaxis equal;

view(3)

hold on

A diagram of a cube

Description automatically generatedA diagram of a cube

Description automatically generated

**Задание 5.** Вращение кубика около одной вершины.

figure('Name','вращение кубика около одной вершины', NumberTitle='off')

% Выбираем вершину, вокруг которой будет вращаться куб

rotation\_point = verticesCube(:,1);

% Делаем вершину центром

center = verticesCube - repmat(rotation\_point, 1, size(verticesCube, 2));

% Поскольку последняя строка не меняет своего значения, мы просто берем

% первые три строки и умножаем их на матрицу вращательного преобразования.

rotatio\_X = [1 0 0;

0 cos(45) sin(45);

0 -sin(45) cos(45)];

rotation\_Y = [cosd(160) 0 sind(160);

0 1 0 ;

-sind(160) 0 cosd(160)];

rotation\_Z= [cosd(50) -sind(50) 0;

sind(50) cosd(50) 0;

0 0 1];

rotatedVertices = rotatio\_X\*rotation\_Y\*rotation\_Z\*center(1:3,:);

%Возвращаем куб в исходную центральную точку

finalVertices = rotatedVertices + rotation\_point(1:3,:);

%добавляем в матрицу последнюю строку со значениями 1

verticesCube5 = [finalVertices; ones(1, size(rotatedVertices, 2))];

DrawShape(verticesCube, facesCube, 'flat');

DrawShape(verticesCube5, facesCube, 'red');

title("вращение кубика около одной вершины")

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

axis equal;

view(3)

hold on

# A graph with different colored cubes Description automatically generatedA graph with different colored cubes Description automatically generatedCamera outline

# A graph with different colored cubes Description automatically generatedЗадание 6. Реализация камеры

figure()

verticesCube\_1 = getTranslated\_Scale(20,-10,5, 2, 2, 4) \* getRotated\_y(25)\*verticesCube;

verticesCube\_2 = getTranslated\_Scale(15, 5, -10, 3, 1, 3)\*getRolated\_z(150)\*getRotated\_x(35)\*verticesCube;

% параметры для камеры

cameraPos = [20 20 15];

cameraTarget =[6 5 3];

cameraUp = [0 1 0];

viewMatrix1 = lookat(cameraPos, cameraTarget, cameraUp) \* verticesCube;

viewMatrix2 = lookat(cameraPos, cameraTarget, cameraUp) \* verticesCube\_1;

viewMatrix3 = lookat(cameraPos, cameraTarget, cameraUp) \* verticesCube\_2;

DrawShape(viewMatrix1, facesCube, 'flat')

DrawShape(viewMatrix2, facesCube, 'red')

DrawShape(viewMatrix3, facesCube, 'blue')

title("Реализация камеры");

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

axis equal

view([0 90])

hold on

% Create a 4x4 view matrix from the target, up, and eye position vectors. This is performing an inverse if the matrix is orthonormalized.

Orientation = [s u f 0;

s u f 0;

s u f 0;

0 0 0 1

];

% Create a 4x4 translation matrix. The eye position is negated which is equivalent to the inverse of the translation matrix. T(v)^-1 == T(-v)

% Finally combine the orientation and translation to compute the final view matrix. Note that the order of multiplication is reversed because the matrices are already inverted.

function viewMatrix = lookat(eye, target, up)

f = normalize(eye - target); % axis z

s = normalize(cross(up,f)); % axis x

u = cross(f,s); % axis y

viewMatrix =[ s(:,1) s(:,2) s(:,3) -dot(s,eye);

u(:,1) u(:,2) u(:,3) -dot(u,eye);

f(:,1) f(:,2) f(:,3) -dot(f,eye);

0 0 0 1

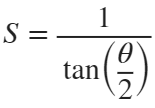
];

end

# A diagram of a scientific experiment Description automatically generated with medium confidenceA diagram of a scientific experiment Description automatically generated with medium confidenceЗадание 7. Реализация перспективы.

***Матрица проекции будет отвечать за:***

- Соотношение сторон (Aspect Ratio): отрегулируйте значения x и y на основе значений ширины и высоты экрана: 

- Поле зрения (Field of view) : отрегулируйте значения x и y в зависимости от угла поля зрения: 

- Нормализация (Normalization): настройте значения x, y и z в диапазоне от -1 до 1:

figure()

fov = 15;

aspect = 1;

n = 3;

f = 2;

Perspective\_1 = createPerspective(fov,aspect,n,f,viewMatrix1);

Perspective\_2 = createPerspective(fov,aspect,n,f,viewMatrix2);

Perspective\_3 = createPerspective(fov,aspect,n,f,viewMatrix3);

title("Реализация перспективы");

xlabel('x','FontSize',10)

ylabel('y','FontSize',10)

zlabel('z','FontSize',10)

view(3)

hold on

% Permission 7 (for 2nd, 3rd and 4th pictures) (1st below)

function Perspective = createPerspective(fovy, aspect, near, far,v)

top = near \* tand((fovy)/2);

bottom = -top;

right = top \* aspect;

left = -right;

%Move the Frustum Apex to the Origin

frustum\_to\_Origin = [

1 0 0 -(left+right)/2;

0 1 0 -(bottom+top)/2;

0 0 1 0

0 0 0 1

];

perspec\_cal = [

near 0 0 0;

0 near 0 0;

0 0 1 0;

0 0 -1 0

];

%Scale the View Window to (-1,1) to (+1,+1)

scale\_wind = [

2/(right-left) 0 0 0;

0 2/(top-bottom) 0 0;

0 0 1 0;

0 0 0 1

];

%Mapping Depth (z values) to (-1,+1)

c1 = 2\*far\*near/(near-far);

c2 = (far+near)/(far-near);

map = [1 0 0 0;

0 1 0 0;

0 0 -c2 c1;

0 0 -1 0];

Perspective = scale\_wind\*perspec\_cal\*map\*frustum\_to\_Origin\*v;

end

# Transformation Matrix

function matrix = getTranslated\_Scale(dx, dy, dz, Sx, Sy, Sz)

matrix = [Sx 0 0 dx;

0 Sy 0 dy;

0 0 Sz dz;

0 0 0 1

];

end

% Rotation

function matrix = getRotated\_x(theta)

matrix = [1 0 0 0;

0 cosd(theta) -sind(theta) 0;

0 sind(theta) cosd(theta) 0;

0 0 0 1

];

end

function matrix = getRotated\_y(theta)

matrix = [ cosd(theta) 0 sind(theta) 0;

0 1 0 0;

-sind(theta) 0 cosd(theta) 0;

0 0 0 1

];

end

function matrix = getRolated\_z(theta)

matrix = [cosd(theta) -sind(theta) 0 0;

sind(theta) cosd(theta) 0 0;

0 0 1 0;

0 0 0 1

];

end

# Задание 7.

% permission 7 (for 1st picture)

projection\_Matrix = perspective(fov, aspect, n, f);

projection\_Matrix\_final\_1 = M\_mul\_v\_project(projection\_Matrix,viewMatrix1);

projection\_Matrix\_final\_2 = M\_mul\_v\_project(projection\_Matrix,viewMatrix2);

projection\_Matrix\_final\_3 = M\_mul\_v\_project(projection\_Matrix,viewMatrix3);

DrawShape(projection\_Matrix\_final\_1, facesCube, 'red')

DrawShape(projection\_Matrix\_final\_2, facesCube, 'blue')

DrawShape(projection\_Matrix\_final\_3, facesCube, 'green')

function projM = perspective(fov, aspect, n, f)

projM = eye(4);

projM(1,1) = 1/aspect\*(tand(fov/2));

projM(2,2) = 1/tand(fov/2);

projM(3,3) = (f+n)/(f-n);

projM(4,3) = f\*n/(f-n);

projM(3,4) = 1;

projM(4,4) = 0;

end

% perform perspective divide with original z-value that is now stored in w

function matrix\_multi\_project = M\_mul\_v\_project(projM, v)

matrix\_multi\_project = projM\*v;

if matrix\_multi\_project(4) ~= 0.0

matrix\_multi\_project(1)= matrix\_multi\_project(1)/matrix\_multi\_project(4);

matrix\_multi\_project(2)= matrix\_multi\_project(2)/matrix\_multi\_project(4);

matrix\_multi\_project(3)= matrix\_multi\_project(3)/matrix\_multi\_project(4);

end

end

# Draw Cube

function DrawShape (vertices , faces , flat )

patch ('Vertices', ( vertices (1:3,:) ./ vertices (4 ,:))', 'Faces', faces ,'FaceVertexCData',hsv(6),'FaceColor', flat, 'facealpha', 0.3)

end

Материалы

1. **Матрица преобразований:** <https://learnopengl.com/Getting-started/Transformations>
2. **Матрица перспективы и ортогональной проекции, матрицы проекций :** <https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/building-basic-perspective-projection-matrix.html>
3. **Перспективные проекции:** <https://learnwebgl.brown37.net/08_projections/projections_perspective.html>