Методические указания к лабораторной работе №2 «Аффинные преобразования» по курсу «Компьютерная графика» для специальностей:

Цель лабораторной работы: Понять принципы работы геометрического конвейера OpenGL c преобразованиями поворота, переноса и масштабирования.

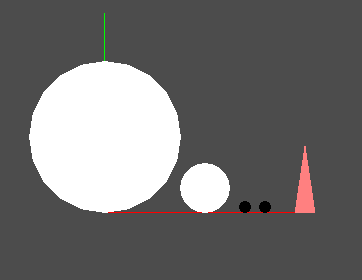
Лабораторная работа основывается на лекциях: «Аффинные преобразования» и «Введение в OpenGL».

Контрольные вопросы по лекционному курсу:

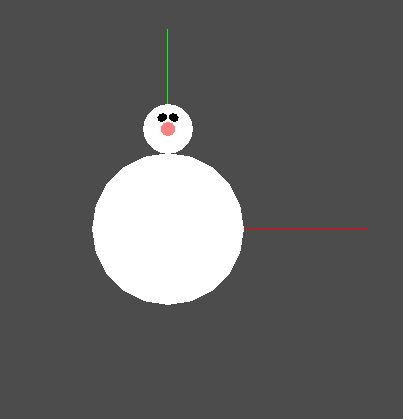
1. Какие аффинные преобразования вы знаете?
2. Что Вы знаете о композиции и коммутативности преобразований?
3. Что делает команда glRotate, сколько у нее параметров и каков их смысл?
4. Как сделать текущую матрицу единичной?
5. Как сохранить текущую матрицу?
6. Что делает команда glTranslate, сколько у нее параметров и каков их смысл?
7. Какое направление поворота считается положительным в OpenGL?
8. Как выбрать тип матрицы, с которой будут работать команды OpenGL?

Для достижения цели необходимо выполнить следующие задания:

1. Расположить составные части модели (снеговика) на оси Х мировой системы координат рис. 1.



1. Собрать модель снеговика и расположить его в начале мировой системы координат рис. 2. Обеспечить вращение модели снеговика вокруг оси Y мировой системы координат.



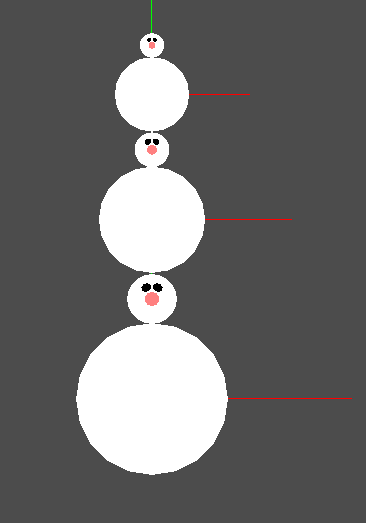
1. Разместить три копии модели вдоль оси Х мировой системы координат, так чтобы центр большой сферы снеговика лежал на оси Х. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Самый большой снеговик размещается в начале мировой системы координат. Снеговики должны касаться друг друга большими сферами, не накладываясь (не пересекаясь). Рис. 3. Обеспечить вращение моделей снеговика вокруг оси Y мировой системы координат.



1. Разместить три копии модели вдоль оси Z мировой системы координат, так чтобы центр большой сферы снеговика лежал на оси Z. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Снеговики должны касаться друг друга большими сферами, не накладываясь (не пересекаясь). Рис. 4. Обеспечить вращение моделей снеговика вокруг оси Y мировой системы координат.



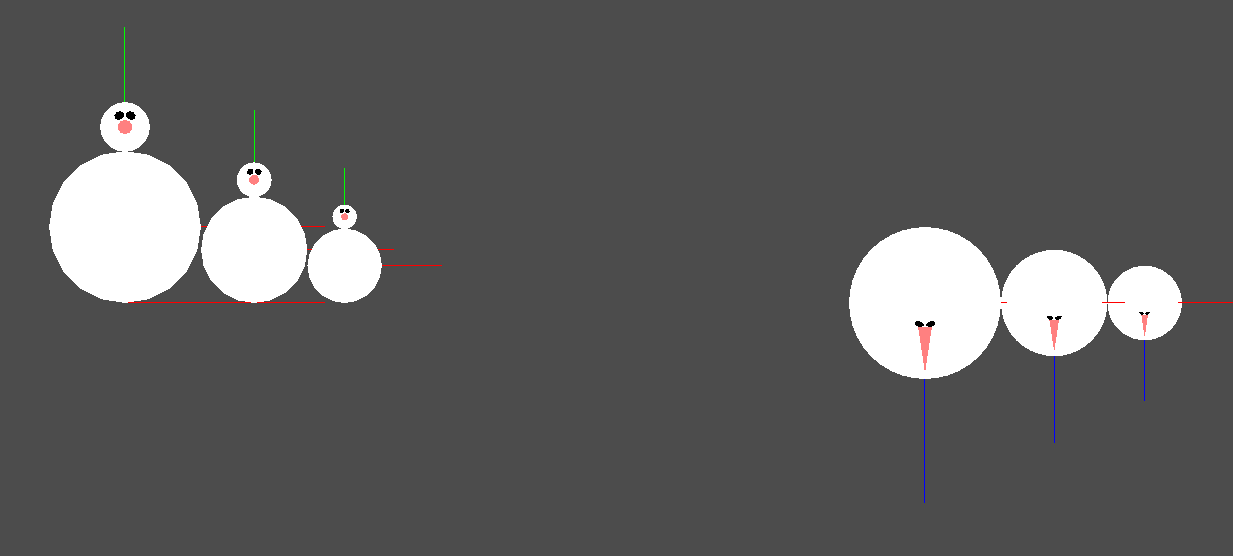
1. Разместить три копии модели вдоль оси Y мировой системы координат, так чтобы центр большой сферы снеговика лежал на оси Y. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Снеговики должны касаться друг друга большими сферами, не накладываясь (не пересекаясь). Рис. 5. Обеспечить вращение моделей снеговика вокруг оси Y мировой системы координат.



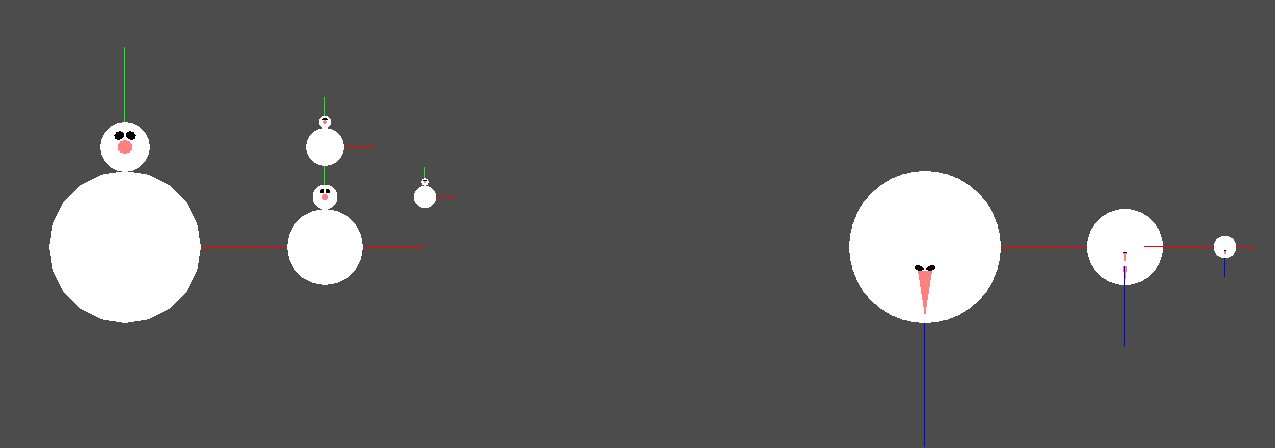
1. Разместить три копии модели вдоль оси, проходящей через точку (1,1,1) мировой системы координат, так чтобы центр большой сферы снеговика лежал на этой оси. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Снеговики должны касаться друг друга большими сферами, не накладываясь (не пересекаясь). Рис. 6. Обеспечить вращение моделей снеговика вокруг указанной выше оси и оси Y мировой системы координат.



1. Разместить три копии модели вдоль оси Х мировой системы координат, так чтобы нижний край большой сферы снеговика касался оси Х. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Проекции Снеговиков на плоскость XOZ должны касаться друг друга большими сферами, не накрадываясь (не пересекаясь). Рис. 7. Обеспечить вращение моделей снеговика вокруг оси Y главной координатной системы среднего снеговика.



1. Разместить три копии модели аналогично предыдущему заданию. Обеспечить вращение каждого снеговика вокруг оси Y его главной координатной системы.
2. Разместить три копии модели аналогично предыдущему заданию. Обеспечить вращение моделей снеговика вокруг оси Y главной координатной системы самого маленького снеговика.
3. Собрать модель снеговика и расположить его в начале мировой системы координат рис. 8. Обеспечить вращение головы снеговика вокруг оси Z мировой системы координат. 
4. Разместить четыре копии модели снеговика. Рис 9. Каждую копию модели, кроме первой масштабировать с коэффициентом 0.7 по всем осям относительно предыдущей копии. Назовем их c1,c2,c3 и c4, в соответствии с размерами. С1 размещается в начале мировой системы координат. Обеспечьте: вращение с2 вокруг оси Y с1; вращение с3 вокруг оси Z c2; вращение С4 вокруг оси Y с2 на уровне головы модели.



Инициализация OpenGL c помощью функций библиотеки GLUT.

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGB); - используем буфер глубины, двойную буферизацию и представление цвета триадой RGB.

glutInitWindowSize(1600, 800);

glutInitWindowPosition(10, 20);

glutCreateWindow("Our first GLUT application!");

glutDisplayFunc(Display); назначаем функцию визуализации

glutSpecialFunc(specialkeys); назначаем функцию обработки нажатия специальных клавиш

Initialize(); дополнительные настройки конвейера

glutMainLoop();

return 0;

}

void Initialize() {

glClearColor(0.3, 0.3, 0.3, 1.0); задаем цвет заливки холста

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0); задаем ортографическую проекцию и видимый объем

glMatrixMode(GL\_MODELVIEW);

glEnable(GL\_DEPTH\_TEST); // включаем тест глубины. Если этого не сделать видимыми будут объекты не расположенные ближе всего к наблюдателю, а созданные последними.

glLoadIdentity();

}

Пример функции обработки нажатия специальных клавиш

void specialkeys(int key, int x, int y) {

if (key == GLUT\_KEY\_LEFT) { rot = rot - 2.0; } else

if (key == GLUT\_KEY\_RIGHT) { rot = rot + 2.0; } else

if (key == GLUT\_KEY\_UP) { alfa = alfa - 2.0; } else

if (key == GLUT\_KEY\_DOWN) { alfa = alfa + 2.0; }

else {

ex = key;

rot = 0;

alfa = 0;

}

glutPostRedisplay(); принудительный вызов функции визуализации

}

Процедура визуализации

Формирование изображения модели в OpenGL начинается с очистки буферов. Обычно это буфер цвета и буфер глубины (glClear). Если используется двойная буферизация, то работа процедуры обязательно завершается переключением буферов (glutSwapBuffers). Так как процедура будет вызываться многократно, необходимо очищать матрицы от накопленных в них преобразований (glLoadIdentity).

В работе необходимо создать две проекции модели, следовательно, мы будем задавать дважды поле вывода (glViewport) вызывать созданную нами процедуру формирования сцены (scene). При инициализации мы задали ортографическую проекцию и вид спереди можно получить без каких-либо дополнительных преобразований. А для создания вида сверху потребуется поворот сцены на 90 градусов относительно оси X мировой системы координат. Процедура drawAxes рисует три единичных отрезка разных цветов вдоль осей координатной системы, зеленый вдоль Y, красный вдоль Х, синий вдоль Z.

void Display() {

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glViewport(0, 0, 800, 800);

glLoadIdentity();

drawAxes();

scene();

glViewport(800, 0, 800, 800);

glLoadIdentity();

glRotatef(90, 1, 0, 0);

drawAxes();

scene();

glutSwapBuffers();

}

Разработка процедуры формирования сцены (scene).

drawSnowMan() - процедура формирующая модель снеговика. Снеговик формируется из сфер и конуса, создаваемых с помощью команд библиотеки GLUT: glutSolidSphere и glutSolidCone см. приложение 1. Процедура будет вызываться многократно при создании сцены, следовательно, она не должна изменять модельно-видовую матрицу. Необходимо воспользоваться стеком матриц и сохранить матрицу модельно-видового преобразования перед началом формирования модели, а после ее создания восстановить матрицую

Рассмотрим пример формирования простой сцены из задания номер 3.

Большой снеговик размещается в начале мировой системы координат. Следовательно, мы просто вызываем процедуру drawSnowMan() передающую модель снеговика на конвейер. Без каких либо преобразований.

Главная координатная система этого снеговика (c1) совпадет с мировой (wc).

c1

c2

wc

Следующий снеговик должен быть уменьшен и сдвинут вдоль оси Х так, что бы касаться первого Расстояния между началом его координатной системы (с2) и мировой, определяется просто: smRad+smRad\*S где smRad – радиус большой сферы снеговика, а S – коэффициент масштабирования. Фрагмент кода формирующй такую сцену может выглядеть так:

drawSnowMan();

glTranslatef(smRad+ smRad\*S, 0.0, 0.0);

glScalef(S, S, S);

drawSnowMan();

Таким образом, из мировой системы координат мы перешли в систему координат с2.

Давайте еще раз рассмотрим выполняемые преобразования. Перенос выполняется в мировых координатах. Масштабирование в сдвинутых относительно мировых. После масштабирования текущие координаты (с2) сдвинуты и их единыцы уменьшены на S относительно мировых. То есть, перенос на единицу, в этих координатах, равен переносу на 1\*S в мировых координатах. Следовательно, перенос на smRad+ smRad\*S в мировых координатах можно заменить на два переноса: на smRad в мировых и на smRad в с2.

drawSnowMan();

glTranslatef(smRad, 0.0, 0.0);

glScalef(S, S, S);

glTranslatef(smRad, 0.0, 0.0);

drawSnowMan();

Разместим третьего снеговика. Его координатная система (с3) смещена относительно мировой на величину равную smRad+ 2\*smRad\*S++ smRad\*S\*S. А смещение относительно с2 равно smRad+ smRad\*S. Разобьем это смещение на два, так как мы это только что делали. Итоговый фрагмент кода формирования сцены из трех снеговиков следующий:

drawSnowMan();

glTranslatef(smRad, 0.0, 0.0);

glScalef(S, S, S);

glTranslatef(smRad, 0.0, 0.0);

drawSnowMan();

glTranslatef(smRad , 0.0, 0.0);

glScalef(S, S, S);

glTranslatef(smRad , 0.0, 0.0);

drawSnowMan();

c1

c2

wc

c3

Реализация преобразований поворота

В OpenGL повороты производятся вокруг векторов привязанных к началу мировых координат. Добавление в программу команды, glRotatetf(alfa,0.0,1.0,0.0) приведет к повороту сцены вокруг оси Y мировой системы координат на угол, заданный переменной alfa. Так как значение этой переменной меняется при нажатии на кнопки клавиатуры, то мы можем сказать, что обеспечили вращение сцены. Ось Y главной координатной системы большого снеговика (с1) совпадает с осью Y мировой системы координат, следовательно мы обеспечили вращение сцены вокруг с1.

glRotatetf(alfa,0.0,1.0,0.0)

drawSnowMan();

glTranslatef(smRad, 0.0, 0.0);

glScalef(S, S, S);

glTranslatef(smRad, 0.0, 0.0);

drawSnowMan();

glTranslatef(smRad , 0.0, 0.0);

glScalef(S, S, S);

glTranslatef(smRad , 0.0, 0.0);

drawSnowMan();

Вращение сцены вокруг оси Y главной координатной системы маленького снеговика (с3) реализуется несколько сложнее. Необходимо выполнить сдвиг по оси X мировой системы координат, такой что, ось Y c3 совпадет с осью Y wc. Затем задать поворот на требуемый угол вокруг Y wc, а потом сдвиг, противоположенный выполненному вначале. Величина сдвига определится как smRad+ 2\*smRad\*S+ smRad\*S\*S

glTranslatef((smRad + 2\*smRad \* S+ smRad \* S\*S ), 0.0, 0.0);

glRotatef(ro, 0, 1, 0);

glTranslatef(-(smRad + 2\*smRad \* S+ smRad \* S\*S), 0.0, 0.0);

glRotatetf(alfa,0.0,1.0,0.0)

drawSnowMan();

glTranslatef(smRad, 0.0, 0.0);

…

Приложение 1. Команды построения примитивов библиотеки GLUT.

Команды, проводящие построение примитивов из библиотеки GLUT, реализованы через стандартные примитивы OpenGL и GLU. Для построения нужного примитива достаточно произвести вызов соответствующей команды.

void **glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks)

void **glutWireSphere**(GLdouble radius, GLint slices, GLint stacks)

Команда glutSolidSphere() строит сферу, а glutWireSphere() - каркас сферы радиусом radius. Параметры slices и stacks определяют количество элементов (разбиений) по осям при аппроксимации поверхности плоскими многоугольниками.

void **glutSolidCube**(GLdouble size)

void **glutWireCube**(GLdouble size)

Эти команды строят куб или каркас куба с центром в начале координат и длиной ребра size.

void **glutSolidCone**(GLdouble base, GLdouble height, GLint slices, GLint stacks)

void **glutWireCone**(GLdouble base, GLdouble height, GLint slices, GLint stacks)

Эти команды строят конус или его каркас высотой height и радиусом основания base, расположенный вдоль оси z. Основание находится в плоскости z=0. Остальные параметры имеют тот же смысл, что и в предыдущих командах.

void **glutSolidTorus**(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings)

void **glutWireTorus**(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings)

Эти команды строят тор или его каркас в плоскости z=0. Внутренний и внешний радиусы задаются параметрами innerRadius, outerRadius. Параметр nsides задает число сторон в кольцах, составляющих ортогональное сечение тора, а rings- число радиальных разбиений тора.

void **glutSolidTetrahedron**(void)

void **glutWireTetrahedron** (void)

Эти команды строят тетраэдр (правильную треугольную пирамиду) или его каркас, при этом радиус описанной сферы вокруг него равен 1.

void **glutSolidOctahedron**(void)

void **glutWireOctahedron**(void)

Эти команды строят октаэдр или его каркас, радиус описанной вокруг него сферы равен 1.

void **glutSolidDodecahedron**(void)

void **glutWireDodecahedron**(void)

Эти команды строят додекаэдр или его каркас, радиус описанной вокруг него сферы равен квадратному корню из трех.

void **glutSolidIcosahedron**(void)

void **glutWireIcosahedron**(void)

Эти команды строят икосаэдр или его каркас, радиус описанной вокруг него сферы равен 1.

Итак, для начала скачайте **glut**. Распакуйте архив и выполните следующие действия:

1. Скопируйте в папку System вашей Windows файл из этого архива **glut32.dll.**

2. Найдите в каталоге, в который вы установили вашу VC++ вложенные директории include\Gl и lib.

У меня это примерно так: D\Program Files\Microsoft Visual Studio 9.0\VC\include\GL и

D:\Program Files\Microsoft Visual Studio 9.0\VC\lib

3. Скопируйте из архива файлы **glut 32. lib**в Program Files\Microsoft Visual Studio 9.0\VC\lib и **glut.h**

в Program Files\Microsoft Visual Studio 9.0\VC\include\GL