Лабораторная работа №7 «Кривые и поверхности».

Поверхности второго порядка предоставляют встроенную поддержку простых поверхностей, которые легко смоделировать с помощью алгебраических уравнений. Однако, если нет алгебраическою уравнения, которое ее описывает, то построение заметно усложняется. Данная нетривиальная задача заключается в том, чтобы в обратном порядке пройти процесс вывода поверхности на экран, начав с ее визуализированного вида и найдя описывающие его полиномы второго или третьего порядка Принятие строгого математического подхода является решением долгим и подверженным ошибкам (даже сели вы привлечете на помощь компьютер), а от мысли сделать все вручную лучше отказаться сразу же

Поняв эту фундаментальную особенность искусства компьютерной трафики, дизайнер автомобилей "Рено" Пьер Безье (Pierre Bezier) в 1970-х создал набор математических моделей, которые могли представлять кривые и поверхности, требуя задания лишь небольшого набора контрольных точек Помимо упрощения представления криволинейных поверхностей эти модели облегчили интерактивную настройку формы кривой или поверхности

Вскоре возникли новые типы кривых и поверхностей. Математика этого трюка не сложнее манипуляций с матрицами, и понять ее интуитивно довольно просто.

Параметрическое представление

Кривая имеет начальную точку, длину и конечную точку. В действительности кривая — это просто линия, которая проходит по трехмерному пространству С другой стороны, поверхность имеет ширину и высоту, следовательно, является областью в пространстве Вначале мы научимся рисовать гладкие кривые в трехмерном пространстве, а затем расширим эту концепцию на поверхности.

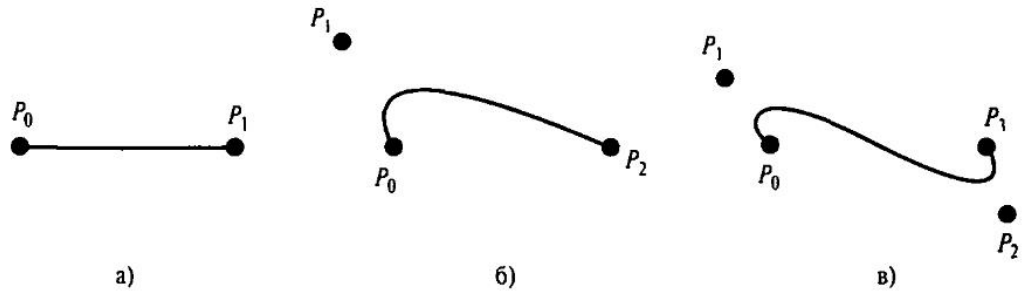

Рис.7.1.Влияние контрольных точек на форму кривой

Контрольные точки

Кривая представляется несколькими контрольными точками, влияющими на форму этой кривой. Для кривой Безье первая и последняя контрольные точки являются частью кривой Другие контрольные точки действуют как магниты, притягивая к себе кривую. Иллюстрация данной концепции для разного числа контрольных точек приведена на рисунке 7.1.

Порядок кривой представляется числом контрольных точек, используемых для описания ее формы *Степень* кривой на единицу меньшее ее порядка. Математическое значение данных терминов связано с параметрическими уравнениями, которые точно описывают кривую, порядком называется число коэффициентов в уравнении, а степенью - наибольший показатель степени параметра в уравнениях.

Кривая на рис. 7.1, *6* называется *квадратичной* (степени 2), а кривая на рис. 7.1, в- *кубической* (степени 3) Кубические кривые наиболее распространены. Теоретически можно определить кривую любого порядка, но чем выше порядок кривой, тем менее контролируемы ее осцилляции, кроме того, кривые высоких порядков сильно меняются при незначительном изменении контрольных точек.

Непрерывность

Если поместить рядом две кривых, создав одну общую точку (именуемую *точкой прерывания),* они сформируют *кусочно-гладкую* кривую. *Непрерывность* данных кривых в точке прерывания описывает, насколько гладким является переход между фрагментами. Существует четыре категории непрерывности, разрыв, позиционная непрерывность (CO), непрерывность касательной (C1) и кривизны (C2).

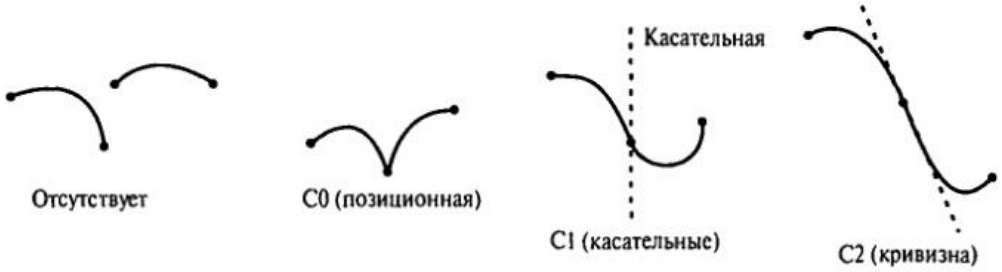

Рис.7.2.Непрерывность кусочно-гладких кривых

Как видно из рис 7.2, если кривые вообще не встречаются, непрерывность отсутствует Позиционная непрерывность достигается в том случае, когда кривые по крайней мере встречаются и имеют общую конечную точку. Непрерывность касательной присутствует, если обе кривых имеют одинаковую касательную в точке прерывания Наконец, непрерывность кривизны означает, что обе кривых также имеют одинаковую скорость изменения в точке прерывания (отсюда и более плавный переход)

При сборке сложных поверхностей или кривых из множества кусочков обычно стараются достичь непрерывности касательной или кривизны. Непрерывность легко получить, правильно выбрав значения некоторых параметров процесса генерации кривых и поверхностей.

Функции оценки

OpenGL содержит несколько функций, облегчающих изображение кривых и поверхностей Безьс. Чтобы нарисовать их, задаются контрольные точки и области изменения параметров u и v. Затем, вызывая подходящую функцию оценки (evaluator), OpenGL генерирует точки, образующие кривую или поверхность. Рассмотрим вначале двухмерный пример кривой Безье, а затем расширим его на три измерения, создав поверхность Безье.

Двухмерная кривая

Начать обучение лучше всего с примера, разбирая его по строчкам Для этого в листинге 7.1 приведен код из программы BEZIER:

Листинг 7.1 Код программы BEZIER, отвечающей за рисование кривой Безье с четырьмя контрольными точками.

```
// Число   контрольных  точек   этой  кривой
GLint nNumPoints = 4;

GLfloat ctrlPoints[4][3]= {{  -4.0f, 0.0f, 0.0f},     //  Конечная точка
       { -3.0f, 4.0f, 0.0f},      //  Контрольная точка
       {  3.0f, -4.0f, 0.0f},     //  Контрольная точка
       {  4.0f, 0.0f, 0.0f }};    //  Конечная точка

//  Эта  функция  используется для  наложения   контрольных  точек  на   кривую
void DrawPoints(void)
    {
    int i;    // Переменная-счетчик

    //   Размер точки   устанавливает  больше,   чтобы  сделать ее   заметнее
    glPointSize(5.0f);

    //  Последовательно  проходятся   все  контрольные  точки данного  примера
    glBegin(GL_POINTS);
```

```
        for(i = 0; i < nNumPoints; i++)
           glVertex2fv(ctrlPoints[i]);
    glEnd();
    }

// Вызывается для рисования сцены
void RenderScene(void)
    {
    int i;
    // Очищает окно текущим цветом очистки
    glClear(GL_COLOR_BUFFER_BIT);

    // Настройка кривых Безье
   // В действительности эти команды нужно вызвать один раз,
   // и их можно перенести в функции установки
    glMap1f(GL_MAP1_VERTEX_3,     // Тип генерируемых данных
    0.0f,                         // Нижняя граница u
    100.0f,                       // Верхняя граница u
    3,                            // Расстояние между точками данных
    nNumPoints,                   // Число контрольных точек
    &ctrlPoints[0][0]);           // Массив контрольных точек

    // Активизируется функция оценки
    glEnable(GL_MAP1_VERTEX_3);

       // Точки соединяются ломаной линией
       glBegin(GL_LINE_STRIP);
              for(i = 0; i <= 100; i++)
                     {
                     // Оценивается кривая в этой точке
                     glEvalCoord1f((GLfloat) i);
             }
    glEnd();

    // Рисуются контрольные точки
    DrawPoints();
    // Выводит команды рисования из стека
    glutSwapBuffers();
    }

 // Эта функция выполняет необходимую инициализацию в контексте визуализации
void SetupRC()
    {
    // Окно очищается до белого цвета
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f );

    // Рисуем синим цветом
    glColor3f(0.0f, 0.0f, 1.0f);
    }

// Устанавливается двухмерная проекция
void ChangeSize(int w, int h)
       {
       // Предотвращает деление на нуль
       if(h == 0)
              h = 1;
       // Размер поля просмотра устанавливается равным размеру окна
    glViewport(0, 0, w, h);
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       gluOrtho2D(-10.0f, 10.0f, -10.0f, 10.0f);
       // Обновления матрицы наблюдения модели
       glMatrixMode(GL_MODELVIEW);
       glLoadIdentity();
       }

int main(int argc, char* argv[])
       {
```

```
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(800, 600);
   glutCreateWindow("2D Bezier Curve");
   glutReshapeFunc(ChangeSize);
   glutDisplayFunc(RenderScene);
   SetupRC();
   glutMainLoop();

   return 0;
   }
```

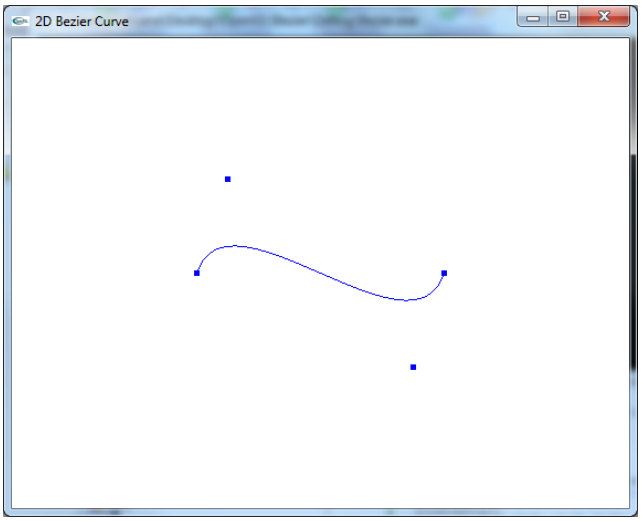

Рис.7.3.Результат выполнения программы BEZIER

Вначале функция RenderScene вызывает glMaplf (после очистки экрана) для отображения кривой. Первый параметр функции glMaplf (GL_MAP1_VERTEX_3) устанавливает функцию оценки, генерирующую тройки координат вершин (x, y и z). Кроме того, можно указать функции оценки сгенерировать другие значения - текстурные координаты и коды цвета.

Следующие два параметра задают нижнюю и верхнюю границы параметра *u* данной кривой. Нижнее значение определяет первую точку кривой, а верхнее — последнюю. Все промежуточные значения соответствуют другим точкам кривой. В данном случае мы задали диапазон 0-100

Четвертый параметр функции glMaplf задает число значений типа float между вершинами в массиве контрольных точек. Поскольку каждая вершина задается тремя значениями (x, y и z), эта величина равна 3. Подобная гибкость позволяет помещать контрольные точки в произвольную структуру данных при условии их размещения с правильным интервалом

Последний параметр является указателем на буфер, который содержит контрольные точки, определяющие кривую. В данном случае мы передали указатель на первый элемент массива. Создав отображение кривой, мы активизировали схему оценки, позволяющую использовать данное отображение. Эта возможность поддерживается с помощью переменной состояния, а чтобы активизировать функцию оценки, дающую контрольные точки вдоль кривой, требуется всего лишь вызывать такую функцию.

Функция glEvalCoordlf принимает один аргумент: параметрическое значение, представляющее точку на кривой. Затем эта функция вычисляет кривую при указанном значении и вызывает glVertex с параметрами найденной точки. Последовательно проходя область определения кривой и вызывая glEvalCoord для нахождения вершин, рисуем кривую, соединяя отрезками найденные точки.

Оценка кривой

OpenGL позволяет существенно облегчить даже описанную выше удобную схему. Создадим с помощью функции glMapGrid сетку, равномерную в области определения *u* (параметрического аргумента кривой). Затем вызовем glEvalMesh, чтобы "соединить точки" с

помощью заданного примитива (GL_LINE или GL_P0INTS). Таким образом, вызываются такие две функции:

```
       // Для отображения в сетку применяется функции  высокого  уровня, // затем вычисляется
все сразу
// Отображается сетка  100 точек от 0 до  100
glMapGrid1d(100, 0.0,100.0);
// С помощью линий вычисляется  сетка
glEvalMesh1(GL_LINE,0,100);
//Эти вызовы полностью заменяют следующий код:
// Точки соединяются ломаной линией
glBegin(GL_LINE_STRIP);
for(i = 0;   i  <= 100;   i++)
{
// Вычисляется   кривая  в данной точке
glEvalCoord1f((GLfloat)   i);
}
       glEnd();
```

Видно, что данный подход компактнее и эффективнее, но его реальная ценность проявляется при оценке поверхностей, а не кривых.

Трехмерная поверхность

Создание трехмерной поверхности Безье весьма похоже на создание ее двухмерного аналога. Помимо вычисления точек вдоль области определения *u* мы должны вычислить их в области определения *v*. В листинге 7.2 приводится код следующей программы BEZ3D, которая отображает каркасную сетку трехмерной поверхности Безье. Первым изменением по сравнению с предыдущим примером является то, что мы определили еще три набора контрольных точек для поверхности в области определения *v*. Чтобы поверхность была простой, в контрольных точках мы меняли только координату *z*. Таким образом была создана гладкая поверхность, представляющая собой как бы двухмерную поверхность Безье, растянутую вдоль оси z.

Листинг 7.1 Код программы BEZ3D, отвечающей за рисование кривой Безье с четырьмя контрольными точками.

```
// Число   контрольных   точек   этой   кривой
GLint nNumPoints = 3;

GLfloat ctrlPoints[3][3][3]= {{{  -4.0f, 0.0f, 4.0f},
                               { -2.0f, 4.0f, 4.0f},
                               { 4.0f, 0.0f, 4.0f }},

                              {{  -4.0f, 0.0f, 0.0f},
                               { -2.0f, 4.0f, 0.0f},
                               {  4.0f, 0.0f, 0.0f }},

                              {{  -4.0f, 0.0f, -4.0f},
                               { -2.0f, 4.0f, -4.0f},
                               {  4.0f, 0.0f, -4.0f }}};


// Эта функция  применяется для наложение  контрольных  точек на кривую
void DrawPoints(void)
    {
    int i,j;  // Переменные-счетчики

    glPointSize(5.0f);
        // Последовательно  проходятся все  контрольные точки данного примера
    glBegin(GL_POINTS);
    for(i = 0; i < nNumPoints; i++)
        for(j = 0; j < 3; j++)
            glVertex3fv(ctrlPoints[i][j]);
    glEnd();
```

```
    }

// Вызывается для рисования сцены
void RenderScene(void)
    {
    // Очищает окно текущим цветом очистки
    glClear(GL_COLOR_BUFFER_BIT);

    // Записывается  стек матриц проекции модели
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    // Поворачиваем сетку,   чтобы сделать ее виднее
    glRotatef(45.0f, 0.0f, 1.0f, 0.0f);
    glRotatef(60.0f, 1.0f, 0.0f, 0.0f);

   // Настройка поверхности  Беэье
   // Эти  команды нужно вызвать один раз,   и их можно
   // перенести  в функции установки
    glMap2f(GL_MAP2_VERTEX_3,     // Тип генерируемых данных
    0.0f,                         // Нижняя  граница u
    10.0f,                        // Верхняя граница u
    3,                            // Расстояние между точками в данных
    3,                            // Размерность  в направлении u (порядок)
    0.0f,                         // Нижняя  граница  v
    10.0f,                        // Верхняя граница v
    9,                            // Расстояние между точками данных
    3,                            // Размерность в направлении  v  (порядок)
    &ctrlPoints[0][0][0]);        // Массив контрольных  точек

    // Активизируется  функция  оценки
    glEnable(GL_MAP2_VERTEX_3);
    // Для отображения  в сетку применяется функции высокого  уровня, //  затем вычисляется
все сразу
       // Отображается  сетка  10 точек от 0 до 10
    glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);
    // С помощью линий оценивается  сетка
    glEvalMesh2(GL_LINE,0,10,0,10);
    // Рисуются  контрольные точки
    DrawPoints();
    // Восстанавливается матрица  наблюдения модели
    glPopMatrix();
    // Отображается  изображение
    glutSwapBuffers();
    }

void SetupRC()
    {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f );
    glColor3f(0.0f, 0.0f, 1.0f);
    }


void ChangeSize(int w, int h)
    {
    if(h == 0)
        h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10.0f, 10.0f, -10.0f, 10.0f, -10.0f, 10.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    }

int main(int argc, char* argv[])
    {
    glutInit(&argc, argv);
```

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(800, 600);
glutCreateWindow("3D Bezier Surface");
glutReshapeFunc(ChangeSize);
glutDisplayFunc(RenderScene);
SetupRC();
glutMainLoop();

return 0;
}
```

Приведенный сейчас код визуализации также отличается от того, что был ранее. Помимо поворота изображения для получения лучшего визуального эффекта, мы вызываем функцию glMap2f вместо glMaplf. Так мы задаем контрольные точки в двух областях определения (u и *v),* а не в одной (u), как было ранее.

Мы по-прежнему должны задавать верхнюю и нижнюю границы *u,* а также расстояние между точками в области определения *u* (как и ранее, оно равно 3). Однако теперь требуется задать еще верхнюю и нижнюю границы области определения *v.* Расстояние между точками в области определения *v* равно теперь девяти значениям, поскольку у нас трехмерный массив контрольных точек, каждой из которых соответствует в области определения *u* три точки по три значения (3x3 = 9). Затем мы сообщаем glMap2f, сколько точек в направлении *v* задается для каждого шага по *u,* после чего предоставляем указатель на собственно контрольные точки.

Двухмерная функция оценки активизируется аналогично рассмотренной выше одномерной, затем вызывается функция glMapGrid2f, в качестве аргумента которой дается число делений области в направлениях *u* и *v.*

После настройки функции оценки можно вызывать двухмерную *(u* и *v)* версию функции glEvalMesh, с помощью которой вычисляется поверхностная сетка. В данном случае применяются прямые отрезки, и задаются значения областей определения *u* и *v* от 0 до 10.

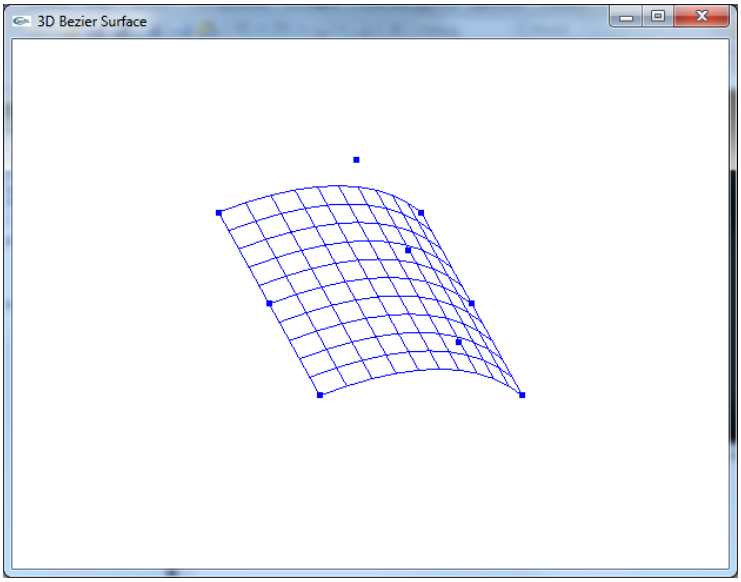

Рис.7.4.Результат выполнения программы BEZ3D

Освещение и векторы нормали

Другой важной особенностью функций оценки является автоматическая генерация нормалей к поверхности Если в коде инициализации заменить

```
// С помощью линий оценивается сетка
glEvalMesh2(GL_LINE,0,10,0,10);
//на
glEvalMesh2{GL_FILL,0,10,0,10);
а затем вызвать функцию
glEnable(GL_AUTO_NORMAL);
```

будет активизировано освещение поверхностей, генерируемых функциями оценки. На рисунке 7.5 показана та же поверхность, что и на рисунке7.4, но с активизированным освещением и включенным автоматическим построением нормалей.

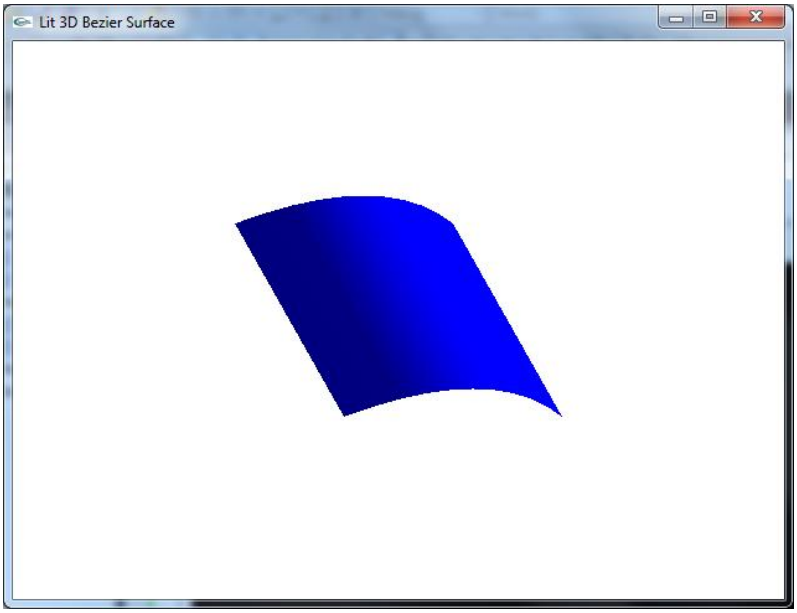

Рис.7.5.Результат выполнения программы BEZ3D_L

Задание:

Нужно создать одну поверхность Безье, заданную как минимум 16-ю контрольными точками, используя функции из библиотеки GL (glMap2[fd], glMapGrid2[fd], glEvalMesh2). Реализовать возможность передвижения (вращения) поверхности для того, чтобы можно было изучить ее со всех сторон.

Два способа реализации:

- *интерактивный* (используя мышку)- перемещение зависит от положения мышки (например, вращение вокруг моментальной оси).

Еще один из возможных способов реализации - это использование специальной панели.

Рис. Пример панели управления.

Если кликать мышкой на левую/правую половину панели, то перемещение (вращение) происходит в положительную/отрицательную сторону. Чем дальше от центра, тем быстрее.

- *неинтерактивный* - перемещение и вращение по заранее заданным траекториям. Не требуется вмешательства пользователя. При этом объект должен хорошо просматриваться со всех сторон!

Дополнительная часть:

Приветствуется реализация перемещения опорных точек с помощью средств GL. Возможность перемещать вдоль оси x, y, z.

Сначала надо дать возможность пользователю выбрать точку. Потом ее переместить. Для выбора точки нужно воспользоваться возможностями GL и GLU по "захвату" части сцены.

Оценка работы:

- Простейший вариант программы без интерактивного манипулирования : **5** баллов
- Простейший вариант программы с интерактивным манипулированием : **7** баллов (+ дополнительные баллы за удобство)
- Перемещение опорных точек : **+5** балла