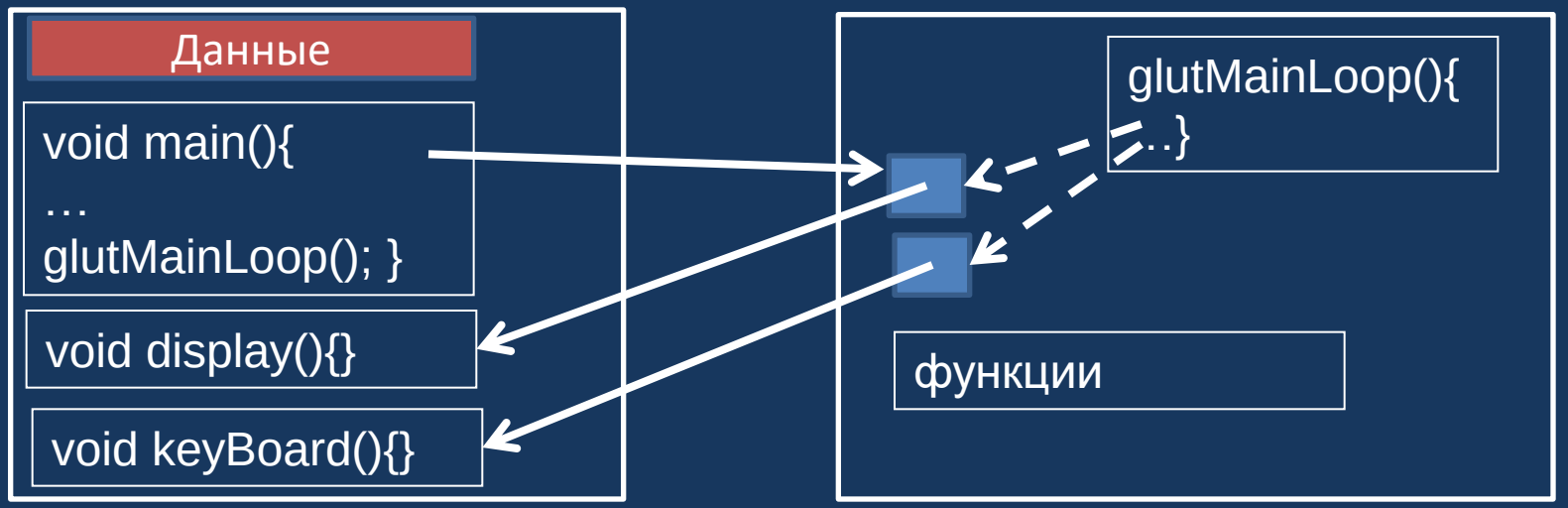
Расчетно-графическая работа. Анимация с использованием 2D-графики

- **2D-графика -** набор примитивов «рисования» на поверхности
  - прямоугольная система координат – «северо-западный» угол - (0,0)
  - примитивы рисования и заливки точек, линий, окружностей, полигонов, строк текста
- библиотека стандартных функций или методов класса Panel или Graphics
- необходимые знания:
  - **подключение библиотеки** к IDE
  - **геометрия** – система координат (прямоугольная, полярная), проекции 3D в 2D, перспектива
  - **динамика** (движение) – координаты, скорость, ускорение, смена направления
  - **физика** – физическая модель – силы, F=ma, ускорение – скорость – координаты (обратная связь) к физической модели = траектория движения
  - **реальное время –** программирование и обработка событий, слежение за временем

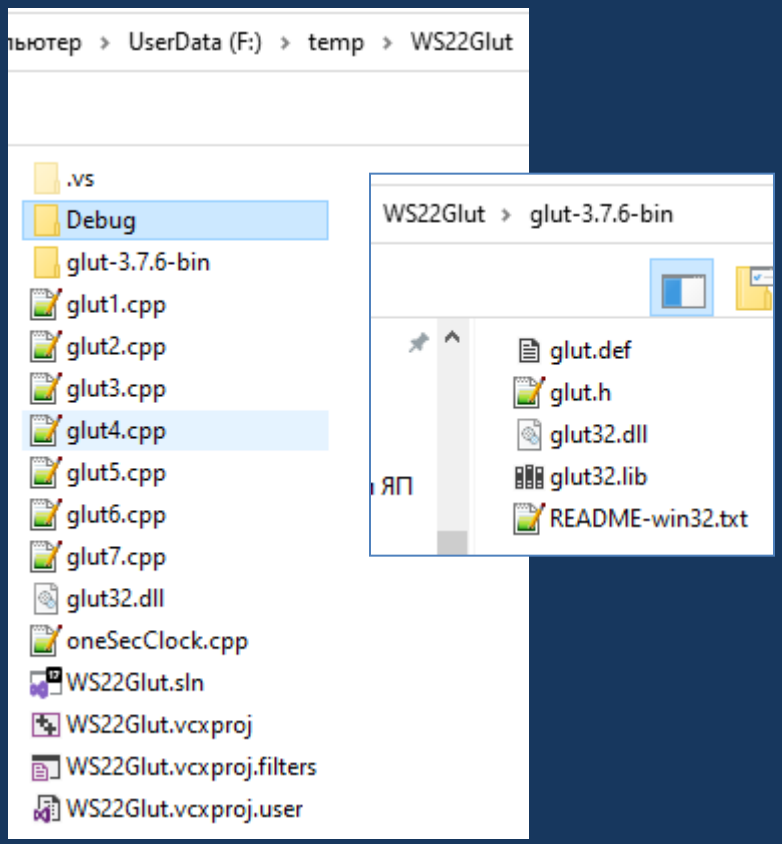
Библиотека Glut (OpenGL)

- 3D – графика
- возможность программирования 2D
- событийная модель - вызов задаваемых функций – обработчиков событий (перерисовка, простой, клавиатура,…)



Настройка проекта Visual Studio

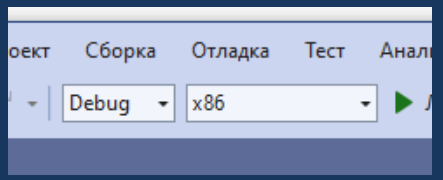
1. Создать пустой проект - консольное приложение
2. В папку проекта скопировать примеры glut1…glut7
3. В папке проекта развернуть архив glut-3.7.6-bin в одноименный каталог
4. Скопировать оттуда glut32.dll в каталог проекта (для запуска из IDE). Для запуска exe-файла скопировать в glut32.dll в каталог, где находится exe-файл

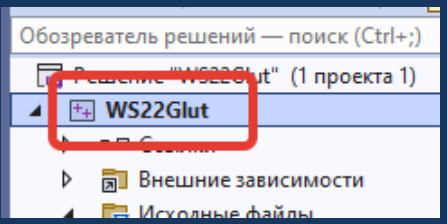


настроенный проект VStutio22Glut.rar

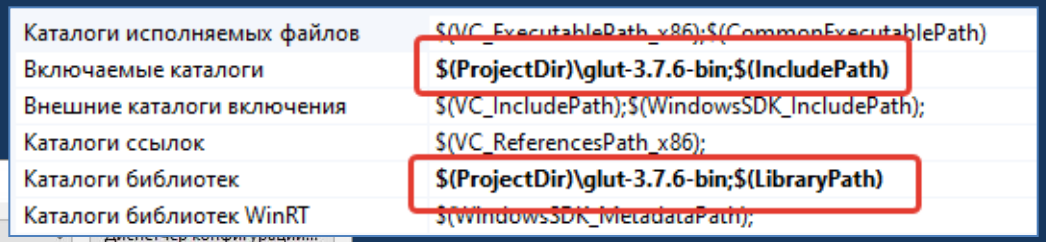
Настройка проекта Visual Studio (2)

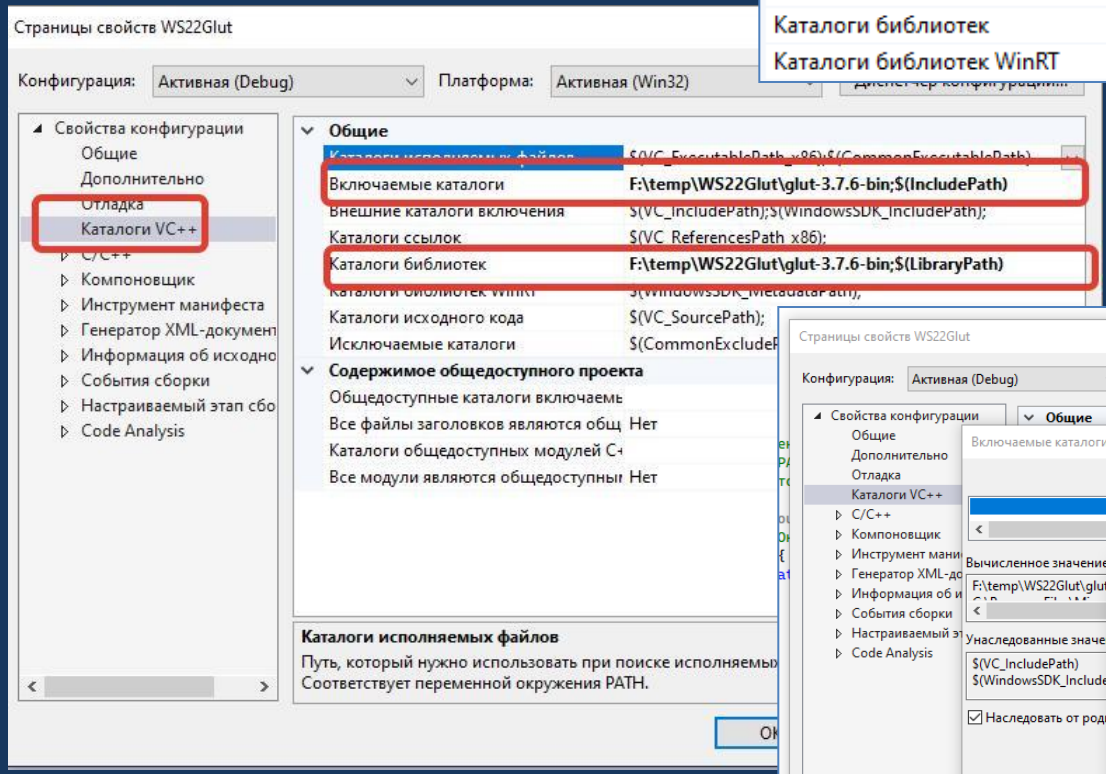
5. Параметры сборки – Debug, x86 (Библиотека win32)
6. Имя проекта – Mouse Right – Свойства – Каталоги C/C++



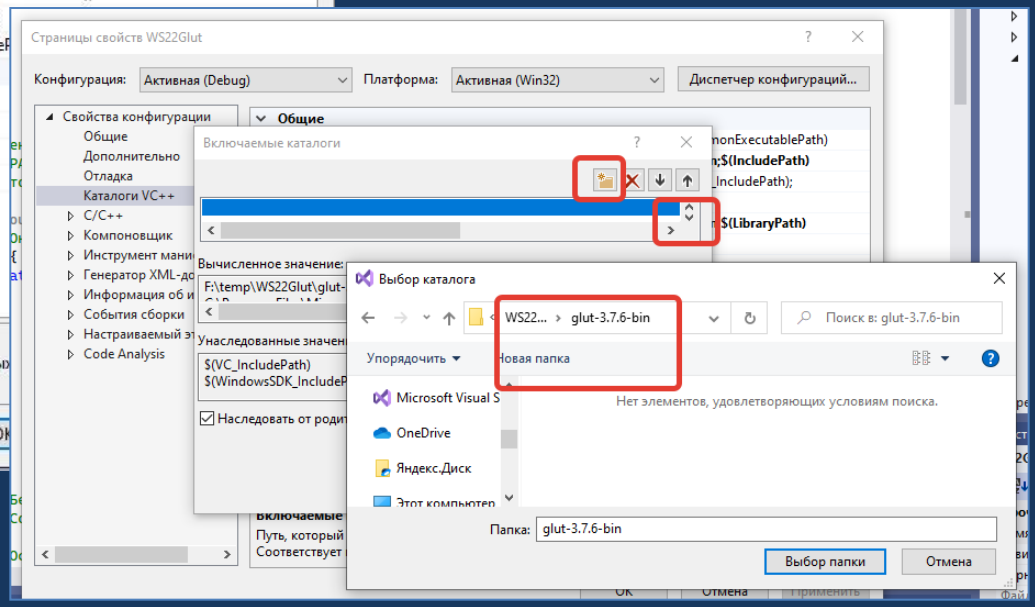


7. Добавить путь glut-3.7.6-bin для include- и lib-каталогов





или так, для настроенного проекта VStutio22Glut.rar



настроенный проект VStutio22Glut.rar

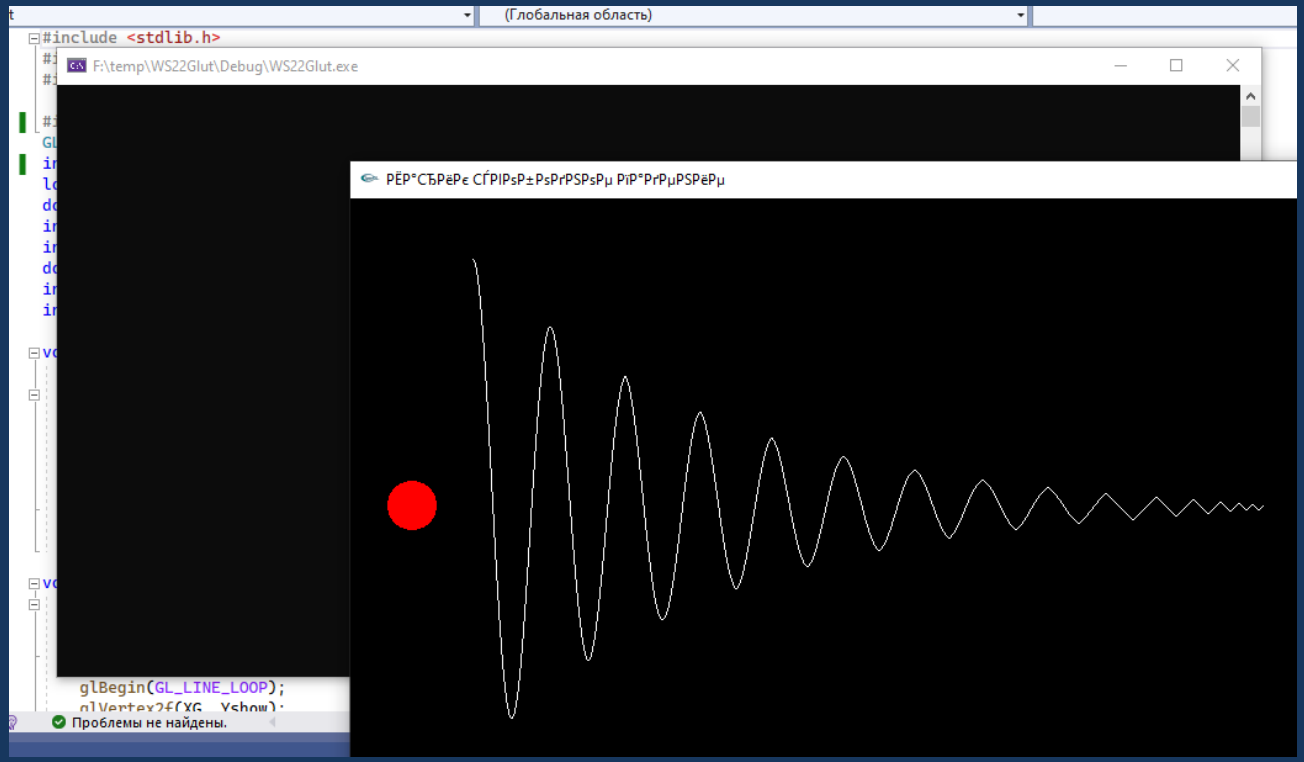
Настройка проекта Visual Studio (3)

8. Добавить файл в проект
9. Исправления кода

```
#include <math.h>

#include <glut.h>
int Width = 500, Height = 500;
int delay=20;              // Не в МКС, а в МС
long time0=0;
double X, Y, Vx, Vy, Ax = 0, Ay = -0.1;
int Xold,Yold,R=20;
int Xshow, Yshow;
double K1 = 0.01, K2 = 0.02;     // Коэффициент у
```

Окна консольного приложения и OpenGL

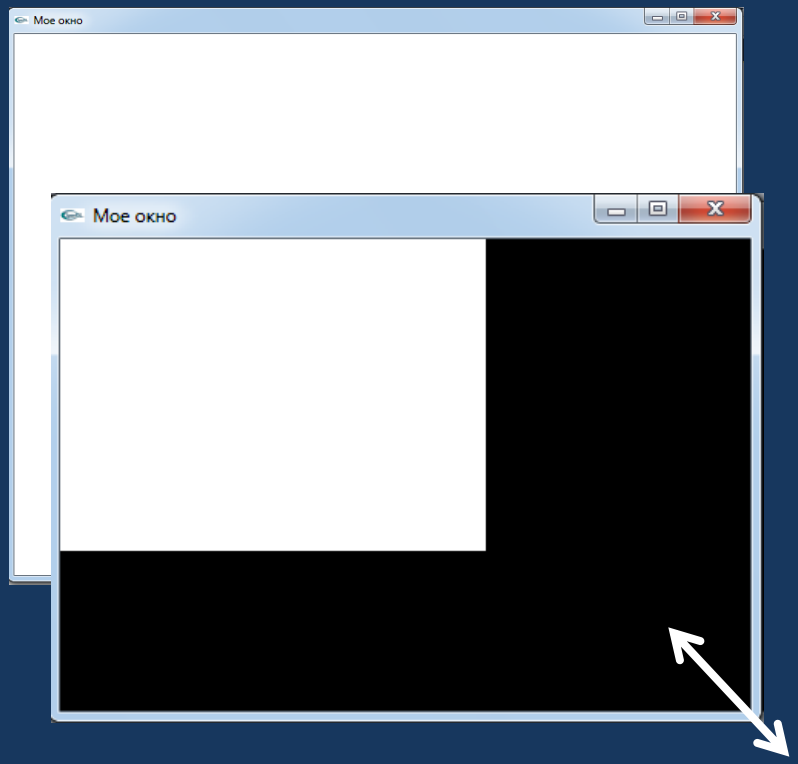


настроенный проект VStutio22Glut.rar

Проверка работоспособности проекта

glut1.cpp – проверка работоспособности (компиляция + сборка + запуск)

- набор функций – обработчиков событий
- настройка обработчиков – передача имени (адреса) функции как ФП
- вызов диспетчера glutMainLoop
- перерисовки НЕТ (изменение
- размеров окна, сворачивание)




```
#include <windows.h>
#include <glut.h>

//Рисование
void Display(){
}
//Обновление экрана
void Reshape(int Width, int higth){
}
//нажатие клавиш
void KeyPress(unsigned char key, int lParam, int rParam){
}
//таймер
void Timer(int lparam){
}

void main(int argv, char *argc[])
{
    glutInit(&argv, argc);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(800, 600);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("Мое окно");
    glutDisplayFunc(Display);
    glutKeyboardFunc(KeyPress);      // ФП - указатель на функцию
    glutReshapeFunc(Reshape);
    glutTimerFunc(0, Timer, 0);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();                  // Диспетчер
}
```

Пример анимации

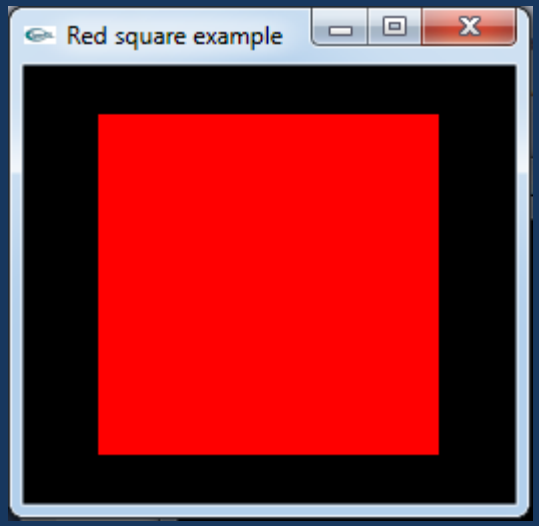
glut2.cpp – пульсирующий квадрат

- события IDLE – отслеживание времени для изменения размеров квадрата (пульсация), инициализация перерисовки
- событие RESHAPE - изменение размеров окна, изменение размеров полотна
- событие рисования –

DISPLAY (система координат **0,0 – левый нижний**)

```
#include <stdlib.h>
#include <time.h>

#include <glut.h>                // Заголовочник  GLUT
GLint Width = 512, Height = 512;// Начальные размеры окна
int CubeSize0 = 200;             // Начальный размер квадрата
int CubeSize = 200;
int dd=3;                        // Приращение размера меняется с + на -
int delay=20;                    // Задержка в МС между последними IDLE
long time0=0;                    // Отметка системного времени
```




```
void main(int argc, char *argv[]){
    time0 = clock();                    // Системное время в МС
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(Width, Height);  // Размер окна
    glutCreateWindow("Red square example");
    glutDisplayFunc(Display);           // Функция отображения
    glutReshapeFunc(Reshape);           // Функция смены размерности окна
    glutKeyboardFunc(Keyboard);         // Функция обработки клавиатуры
    glutIdleFunc(Idle);                 // Функцция простоя IDLE
    glutMainLoop();                     // Main библиотеки
    }
```

Пример анимации

glut2.cpp – пульсирующий квадрат

```
void Display(void){             // Вывод на экран
    int left, right, top, bottom;
    left  = (Width - CubeSize) / 2;
    right = left + CubeSize;
    bottom = (Height - CubeSize) / 2;
    top = bottom + CubeSize;    // Координаты - квадрат ПО СЕРЕДИНЕ
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255,0,0);        // RGB - цвет заливки
    glBegin(GL_QUADS);          // Начало полигона
    glVertex2f(left,bottom);    // Вершина полигона
    glVertex2f(left,top);
    glVertex2f(right,top);
    glVertex2f(right,bottom);
    glEnd();
    glFinish();
    }

void Reshape(GLint w, GLint h){     // Событие - изменение размеров окна
    Width = w;                      // Запомнить новые размеры
    Height = h;
    glViewport(0, 0, w, h);         // Область отображения
    glMatrixMode(GL_PROJECTION);    // Ортографическая проекция
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    }

void Keyboard(unsigned char key, int x, int y){
    #define ESCAPE '\033'   // Событие - символ клавиатуры
    if( key == ESCAPE )     // Клавиша ESCAPE
        exit(0);            // Закрыть программу
    }
```

```
void Idle(){
    if (clock()-time0 < delay)
        return;
    time0=clock();
    if (CubeSize<30 && dd>0 || CubeSize>200 && dd<0){
        dd=-dd;
        }
    CubeSize-=dd;
    glutPostRedisplay();
    }
```

Геометрия 2D

Использование прямоугольной и полярной, абсолютной и относительной систем координат

Пример. Движущийся и вращающийся многоугольник

1. Координаты центра X0,Y0 – движение всей фигуры
2. Вращение – вектор, текущий угол поворота φ, радиус R
3. N вершин многоугольника $\varphi i = \varphi + (2\pi)i/N$
4. Координаты вершин $xi = X0 + R\cos(\varphi i)$
5. Соединить линиями соседние пары (выпуклый) или через одну (звезда)

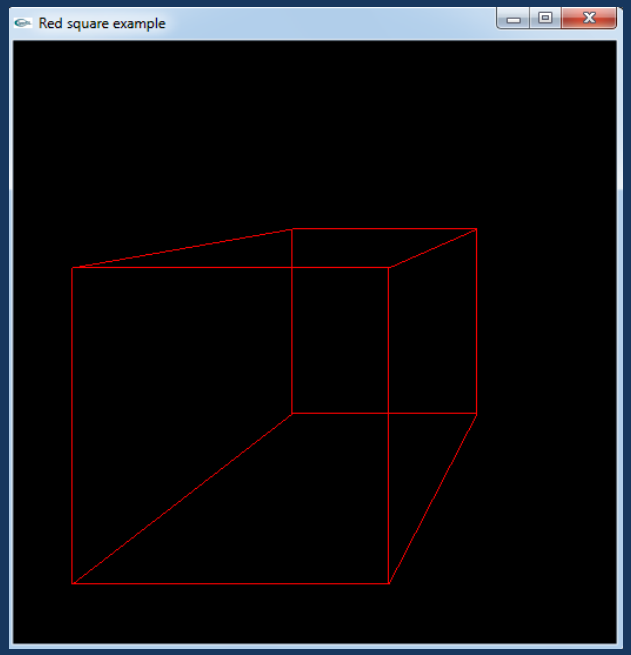
Геометрия 3D

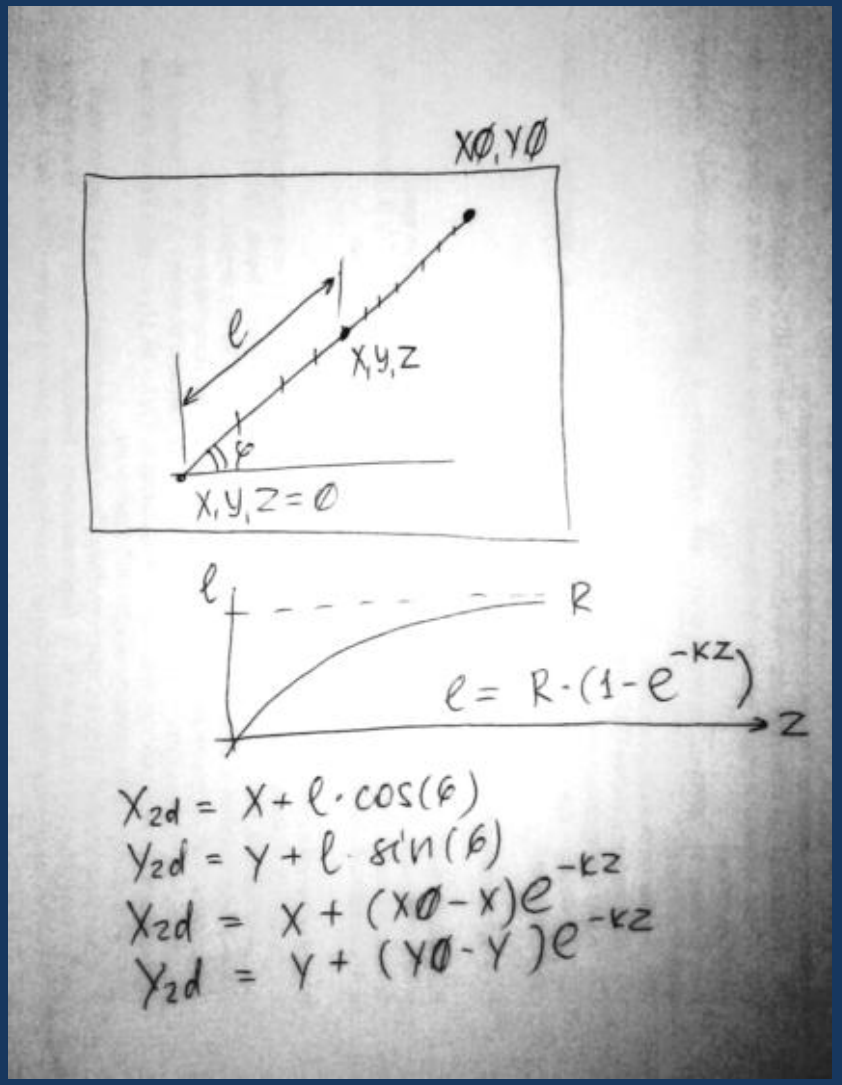
1. использовать функции 3D-графики

2. преобразование объемных координат в плоские (проекция, перспектива) + линейная алгебра

пример: пульсирующий куб

в перспективе (glut7.cpp)





Геометрия 3D

```
#include <glut.h>                // Заголовочник  GLUT
GLint Width = 512, Height = 512;// Начальные размеры окна
int SZ = 200;
int XY0 = 50;
int dd=3;                        // Приращение размера меняется с + на -
int delay=20;                    // Задержка в МС между последними IDLE
long time0=0;                    // Отметка системного времени
int X0 = 500, Y0 = 400;          // Координаты БЕСКОНЕЧНОСТИ
double K = 0.002;                // Сжатие по оси Z (перспектива)

int get3DX(int x, int y, int z){
    int  xx = x + (X0 - x)*(1 - exp(-K*z));
    //printf("x=%d\n", xx);
    return xx;
    }
int get3DY(int x, int y, int z){
    return y + (Y0 - y)*(1 - exp(-K*z));
    }
void line3D(int x1, int y1, int z1, int x2, int y2, int z2){
    glBegin(GL_LINE_LOOP);
    glVertex2f(get3DX(x1, y1, z1), get3DY(x1, y1, z1));
    glVertex2f(get3DX(x2, y2, z2), get3DY(x2, y2, z2));
    glEnd();
    }

void Display(void){              // Вывод на экран
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255,0,0);         // RGB - цвет заливки
    line3D(XY0, XY0, 0, XY0 + SZ, XY0, 0);
    line3D(XY0, XY0, 0, XY0, XY0 + SZ, 0);
    line3D(XY0+SZ, XY0, 0, XY0 + SZ, XY0+SZ, 0);
    line3D(XY0, XY0+SZ, 0, XY0 + SZ, XY0+SZ, 0);
    line3D(XY0, XY0, SZ, XY0 + SZ, XY0, SZ);
    line3D(XY0, XY0, SZ, XY0, XY0 + SZ, SZ);
    line3D(XY0 + SZ, XY0, SZ, XY0 + SZ, XY0 + SZ, SZ);
    line3D(XY0, XY0 + SZ, SZ, XY0 + SZ, XY0 + SZ, SZ);
    line3D(XY0, XY0, 0, XY0, XY0, SZ);
    line3D(XY0, XY0 + SZ, 0, XY0, XY0 + SZ, SZ);
    line3D(XY0 + SZ, XY0, 0, XY0 + SZ, XY0, SZ);
    line3D(XY0 + SZ, XY0 + SZ, 0, XY0 + SZ, XY0 + SZ, SZ);
    glFinish();
    }
```

Динамика. Симуляция закона движения

1. Дискретное пространство – прямоугольная система координат (**пиксель**)
2. Дискретное время – **цикл** (шаг моделирования)
3. Скорость движения – **пискель/цикл**
4. Переменные движения – вещественные

```
double X, Y, Vx, Vy, Ax, Ay;
```

6. Для эффективного отображения при низкой скорости фиксировать изменение целочисленной составляющей координаты

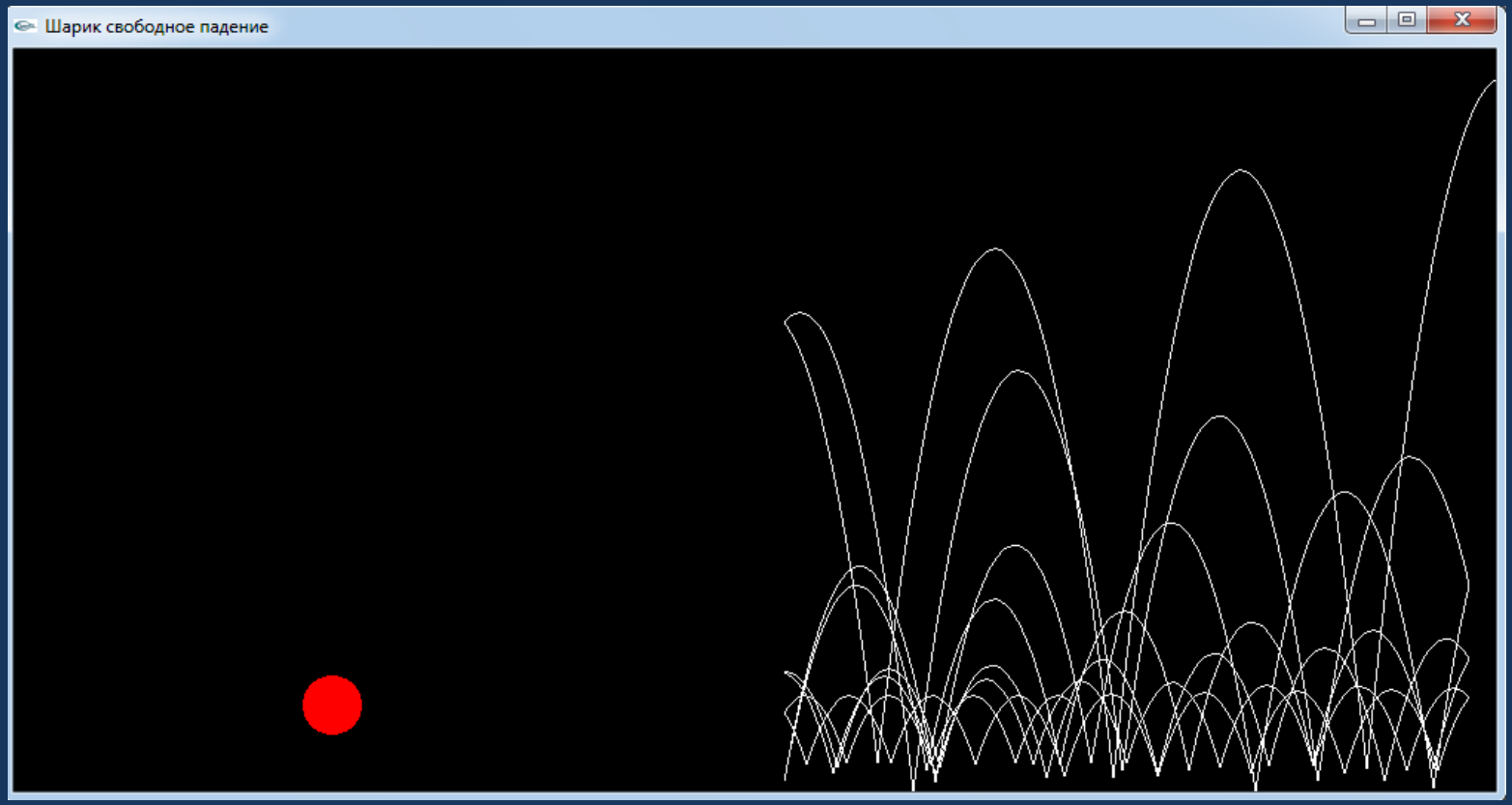
```
int oldX = X, oldY = Y; …. изменение X, Y
if (oldX !=(int)X || oldY!=(int)Y){
        oldX = X; oldY = Y;
        перерисовать по oldX,oldY; }
```

7. Изменение скорости и координаты – интегрирование (суммирование) ускорения и скорости Vx += Ax; X += Xv;
8. Эффективная перерисовка объекта – стереть (рисовать цветом фона), переместить, рисовать основным цветом
9. Движение группы объектов = слежение за всеми в одном событии (иначе, внутренний параллелизм, потоки – Thread (C#, Java))

Динамика. Симуляция закона движения

Пример. Свободно падающий шарик с неупругим отскоком (Glut4.cpp)

- ax = 0 ay = -0.1 (вниз)
- отскок – изменение знака скорости при достижении края
- перерисовка шарика = стирание старого положения и рисование нового (для сохранения трассы)



Динамика. Симуляция закона движения

```
void drawCircle(int x, int y, int r, int amountSegments){
    glBegin(GL_POLYGON);                // Окружность - полигон ВРУЧНУЮ
    for(int i = 0; i < amountSegments; i++){
        float angle = 2.0 * 3.1415926 * float(i) / float(amountSegments);
        float dx = r * cosf(angle);
        float dy = r * sinf(angle);
        glVertex2f(x + dx, y + dy);
        }
    glEnd();
    }

void Display(void){
    // Не стирать, чтобы видеть ТРАССУ
    //glClearColor(0, 0, 0, 1);
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255, 255, 255);          // Белый график
    glBegin(GL_LINE_LOOP);              // Соединить старую и новую точки
    glVertex2f(Xshow + Width/2, Yshow); //
    glVertex2f(Xold + Width/2, Yold);   //
    glEnd();
    glColor3ub(0, 0, 0);
    drawCircle(Xshow, Yshow, R, 50);    // Стереть
    glColor3ub(255, 0, 0);
    drawCircle(Xold, Yold, R, 50);      // Нарисов
    Xshow = Xold; Yshow = Yold;         // Нове ме
    glFlush();
    glFinish();
    }
```

```
void Idle(){    // Задержка и ДИНАМИКА
    if (clock()-time0 < delay)
        return;
    time0=clock();
    X += Vx;            // Интегрирование координат
    Y += Vy;
    Vx += Ax;           // Интегрирование скорости
    Vy += Ay;
    if (Vx<0 && X <= R) Vx = -Vx;   // Отскок от краев
    if (Vx>0 && X > Width/2 - R) Vx = -Vx;
    if (Vy<0 && Y <= R) Vy = -Vy*0.9;
    if (Vy>0 && Y > Height - R) Vy = -Vy;
    if ((int)X != Xold || (int)Y != Yold){
        Xold = X;                   // Перемещение более чем на пиксель
        Yold = Y;
        glutPostRedisplay();        // Инициализировать перерисовку
        }
    }
```

Физика

Основа правдоподобной анимации – моделирование движения в соответствии законами физики

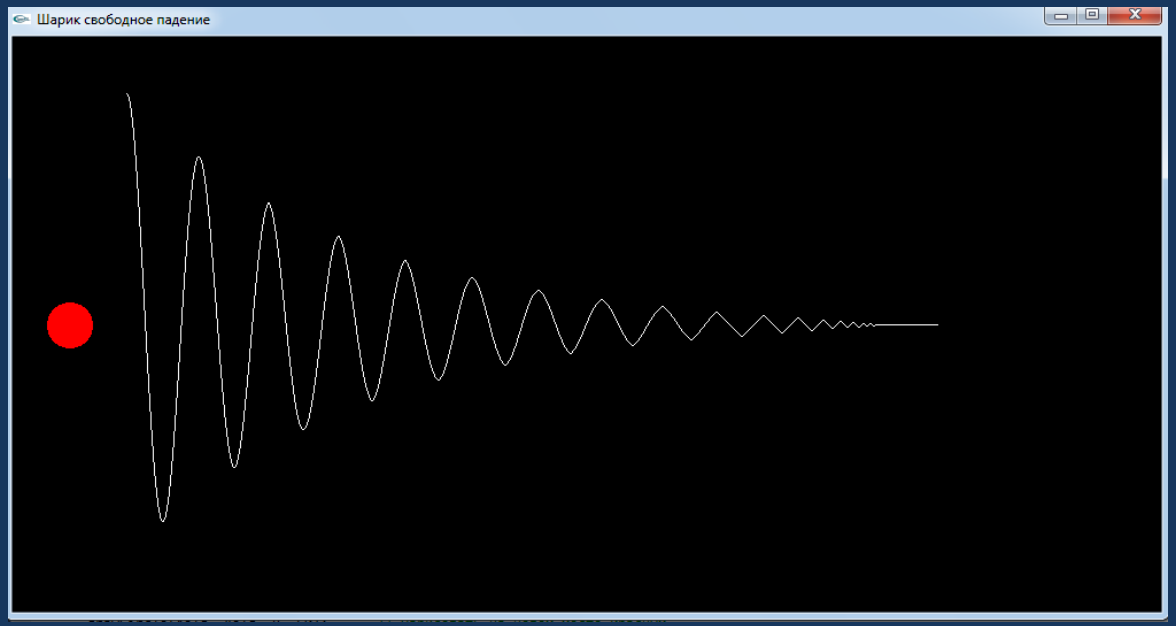
1. Описание закона движения $F= ma$ (в векторной форме), или в координатной $F_x = ma_x$, $F_y = ma_y$
2. Сумма сил F – каждая составляющая может зависеть от координат и скорости, т.е. $F_x(V_x, V_y, x , y)$
3. $a_x = V_x'$, $V_x = x'$ - ускорение = производная скорости, скорость = производная координаты по времени
4. скорость = интеграл от ускорения, координата = интеграл скорости
5. $F_x (x(t)', y(t)' , y(t)) = m\, x(t)''$ – дифференциальное уравнение, связывает неизвестные функции x(t), y(t) с их производными
6. Для блондинок – в точке действуют силы, вызывающие ускорение, которое меняет скорость, которая меняет координаты точки, в которой действуют уже другие значения сил = траектория движения
7. **аналитическое решение –** поиск функции в общем виде, которая удовлетворяет дифференциальным уравнениям
8. **моделирование (симуляция, имитация) –** компьютерное воспроизведение движения в дискретном пространстве - времени

Физика. Затухающие колебания

Пример. Затухающие гармонические колебания (glut5.cpp)

1. $F_y = -K*(y-y0) – K2*V_y = ma_y$ – возвратная сила пружины + сопротивление движению
2. Аналитическое решение $Y(t) = A*\sin(\omega t+\varphi)*\exp(-Kt)$
3. $F_y = -K*(y-y0) = ma_y$ – незатухающие колебания

   $-K*(y(t) – y0) = my(t)''$ – вторая производная от функции равна ей же самой с противоположным знаком



Физика. Затухающие колебания

```
void Display(void){
    // Не стирать, чтобы видеть ТРАССУ
    //glClearColor(0, 0, 0, 1);
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255, 255, 255);          // Белый график
    glBegin(GL_LINE_LOOP);              // Соединить старую и новую точки
    glVertex2f(XG, Yshow);              //
    glVertex2f(XG-1, Yold);             // Ось X - ось времени
    glEnd();
    XG++;
    if (XG == Width) XG = Width / 10;   // Дошли до конца - в начало
    glColor3ub(0, 0, 0);
    drawCircle(Xshow, Yshow, R, 50);
    glColor3ub(255, 0, 0);
    drawCircle(Xold, Yold, R, 50);
    Xshow = Xold; Yshow = Yold;
    glFlush();
    glFinish();
    }
```

```
void Idle(){    // Задержка и ДИНАМИКА
    if (clock()-time0 < delay)
        return;
    time0=clock();              // Ускорение пропорционально растяжению пружины
    Ay = -K1*(Y - Y0) - K2*Vy;  // и скорости движения (оба со знаком - )
    X += Vx;                    // Интегрирование координат
    Y += Vy;
    Vx += Ax;                   // Интегрирование скорости
    Vy += Ay;
    if ((int)X != Xold || (int)Y != Yold){
        Xold = X;                   // Перемещение более чем на пиксель
        Yold = Y;
        glutPostRedisplay();        // Инициализировать перерисовку
        }
    }
```

```
void main(int argc, char *argv[]){
    Y0 = Height / 2;
    Vx = 0;
    Vy = 0;
    Ax = 0;
    X = Xshow = Xold = Width / 20;
    Y = Yshow = Yold = Y0+200;
    XG = Width / 10;
```

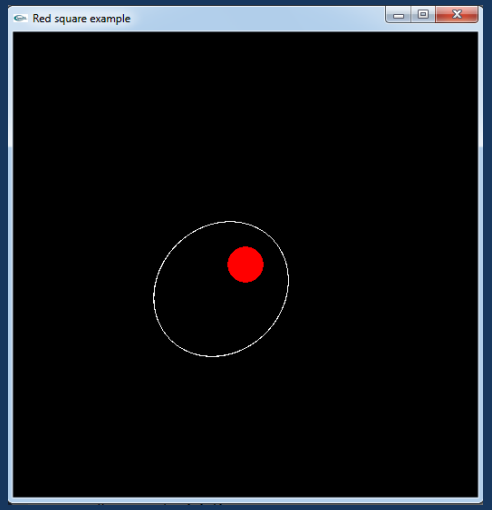
Физика. Движение в гравитационном поле

Пример. Движение в гравитационном поле (glut3.cpp)

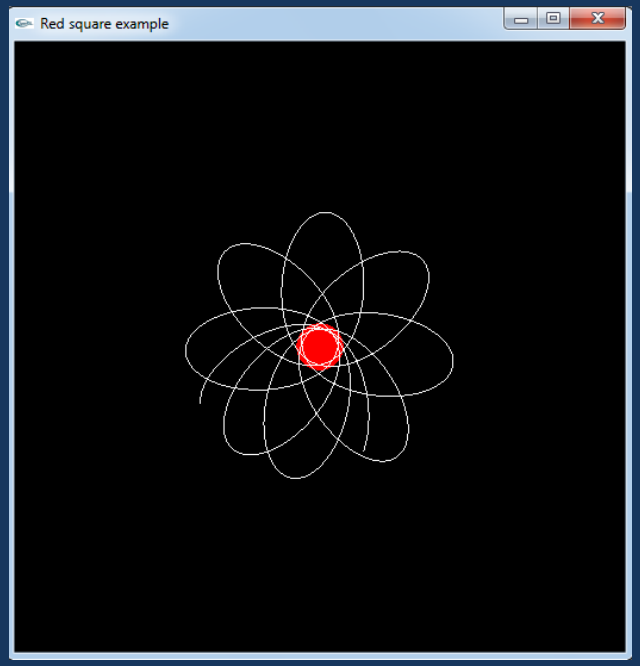
1. Координаты «солнца». Начальные координаты и скорость «небесного тела»
2. Сила тяготения – вектор, F = Kg*m1*m2/R^2 = ma
3. Модуль ускорения a = K / R ^2
4. Ускорение проецируется на ax и ay, скорость и координаты интегрируются

   «Корректировка» закона всемирного тяготения

   POW = 2



POW = 1.75



Физика. Движение в гравитационном поле

```
void Idle(){
    if (clock()-time0 < delay)
        return;
    time0=clock();
    int cl,dd,xx,yy;
    int sx=(xx1<xx0 ? 1 : -1);              // Квадрант НЕБЕСНОГО ТЕЛА
    int sy=(yy1<yy0 ? 1 : -1);              // (знак проекции ускорения)
    r=sqrt(pow(xx1-xx0,2)+pow(yy1-yy0,2));  // Расстояние
    double fi=asin(fabs(yy1-yy0)/r);        // УГОЛ между СОЛЦЕМ и ТЕЛОМ
    double aa=K/(pow(r,POW));               // Модуль ускорения
    ax=sx*aa*cos(fi);                       // Проекции ускорения
    ay=sy*aa*sin(fi);
    vx+=ax;                                 // Интегрирование движения
    vy+=ay;
    xx1+=vx;
    yy1+=vy;
    glutPostRedisplay();
    }

void main(int argc, char *argv[]){
    xx0 = Width / 2;    // Координаты СОЛНЦА
    yy0 = Height/2;
    xx1=xx0-100;        // Координаты НЕБЕСНОГО ТЕЛА
    yy1=yy0-50;
    vx=-0.2;            // Начальная скорость
    vy=1;
    dx=0;
    dy=0;
    K=200;
    POW = 1.75;         // СТЕПЕНЬ закона тяготения
    xold=yold=0;
    time0 = clock();
```

Физика. Реальный маятник

Пример. Реальный маятник (glut6.cpp)

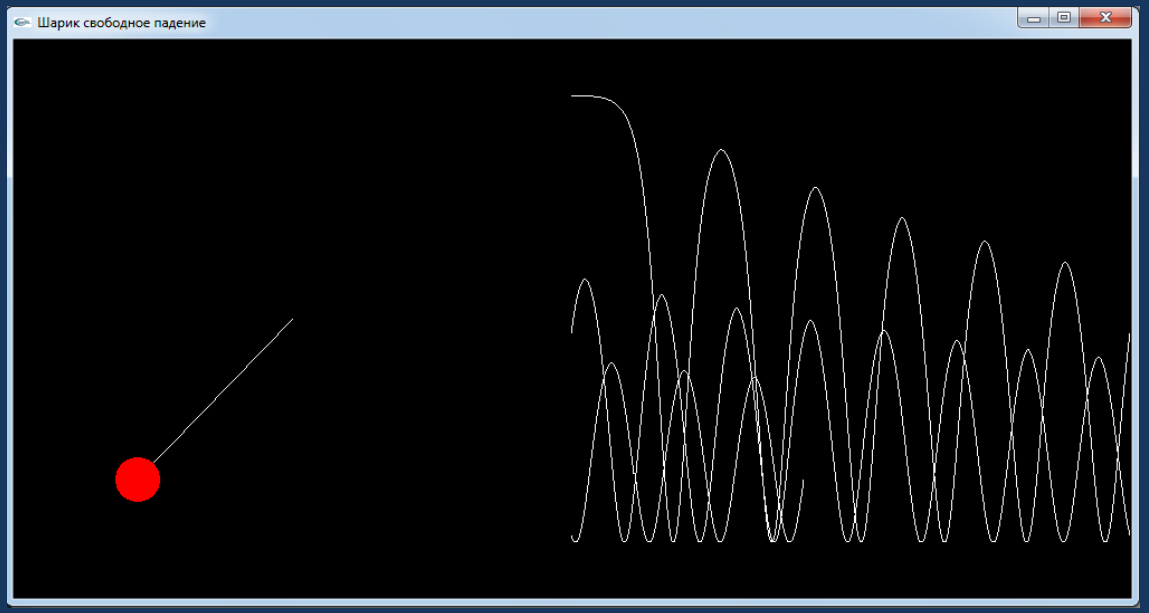
1. Полярная система координат, вращение вокруг своей оси:

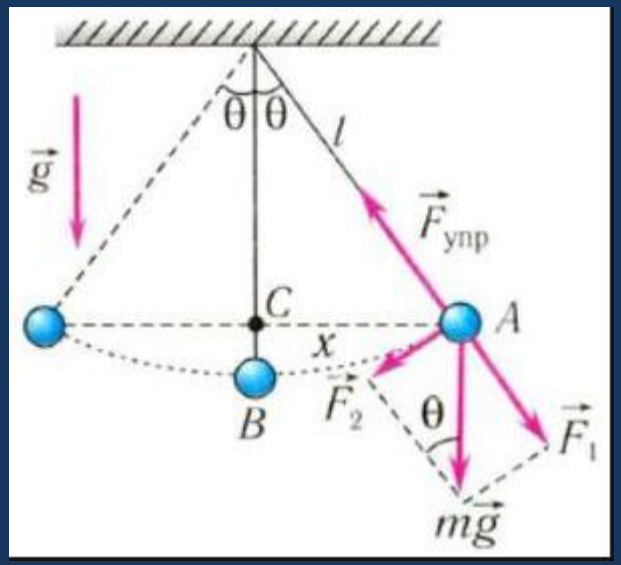
   aω – угловое ускорение, ω – угловая скорость, φ – угол

2. Расстояние R = const

3. Правила интегрирования движения в полярной системе - аналогичны для вращения и радиуса

4. Проекция силы тяжести на ось вращения = - m*G*sin(φ), для малых углов - sin(φ) ≈ φ, уравнение гармонических колебаний





Физика. Реальный маятник

```
#include <glut.h>
GLint Width = 1000, Height = 500;
int delay=20;
long time0=0;
double X, Y;
double AOM, OM, FI;              // Угловое ускорение, скорость и угол поворота
int Xold,Yold;
int Xshow, Yshow;
double K1 = 0.003, K2 = 0.003;   // Коэффициент упругости и затухания
int X0 = 200, Y0 = 250, R=200;   // Центр и длина маятника
int XG;                          // Текущая точка графика (ось времени)
int RM = 20;
```

```
void Display(void){
    // Не стирать, чтобы видеть ТРАССУ
    //glClearColor(0, 0, 0, 1);
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255, 255, 255);
    glBegin(GL_LINE_LOOP);               // Соединить старую и новую точки
    glVertex2f(XG, Yshow);               //
    glVertex2f(XG-1, Yold);              // Ось X - ось времени
    glEnd();
    XG++;
    if (XG == Width) XG = Width / 2;     // Дошли до конца - в начало
    glColor3ub(0, 0, 0);
    glBegin(GL_LINE_LOOP);               // Стереть ось маятника
    glVertex2f(X0, Y0);                  //
    glVertex2f(Xshow, Yshow);
    glEnd();
    glColor3ub(255, 255, 255);           // Нарисовать ось маятника
    glBegin(GL_LINE_LOOP);               //
    glVertex2f(X0, Y0);                  //
    glVertex2f(Xold, Yold);
    glEnd();
    glColor3ub(0, 0, 0);
    drawCircle(Xshow, Yshow, RM, 50);    // Стереть - старый цветом фона
    glColor3ub(255, 0, 0);
    drawCircle(Xold, Yold, RM, 50);      // Нарисовать на новом месте красным
    Xshow = Xold; Yshow = Yold;          // Новое место = старое
    glFlush();
    glFinish();
    }
```

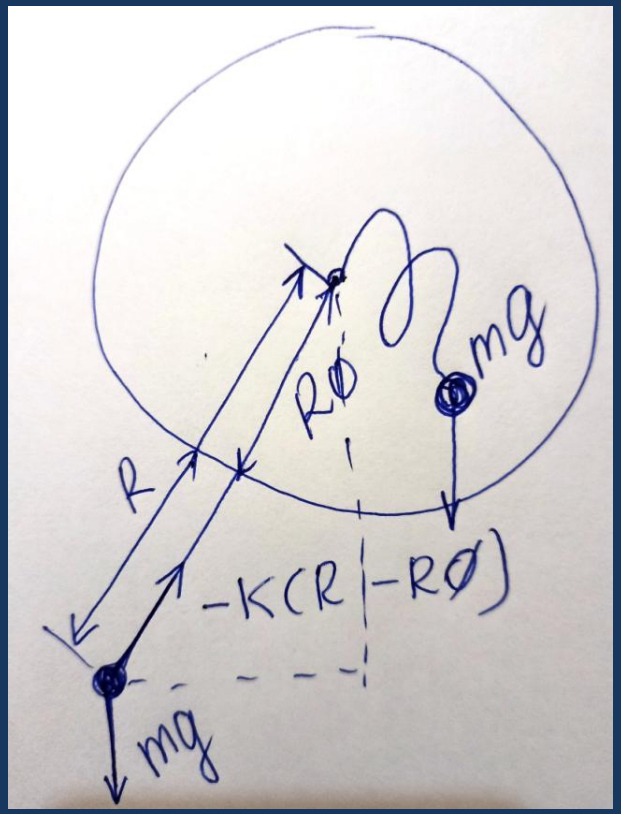
```
void Idle(){    // Задержка и ДИНАМИКА
    if (clock()-time0 < delay)
        return;
    time0=clock();                       // Уско
    AOM = -K1 * sin(FI) - K2 * OM;
    FI += OM;                            // Инте
    OM += AOM;
    X = X0 + R * sin(FI);                // Пере
    Y = Y0 - R * cos(FI);                //
    if ((int)X != Xold || (int)Y != Yold){
        Xold = X;                        // Пере
        Yold = Y;
        glutPostRedisplay();             // Иниц
        }
    }
```

Замечания по моделированию

Маятник на пружине (штоке). Полярная система координат, вращение вокруг своей оси: aω – угловое ускорение, ω – угловая скорость, ϕ – угол, R – текущий радиус, ar - центростремительное ускорение

Проекции сил на ось вращения = - m*G*sin(ϕ)
Проекции сил на радиальную ось m*G*cos(ϕ)-k(R-R0)

«Шарик на резинке»



«Насекомое и лампочка»