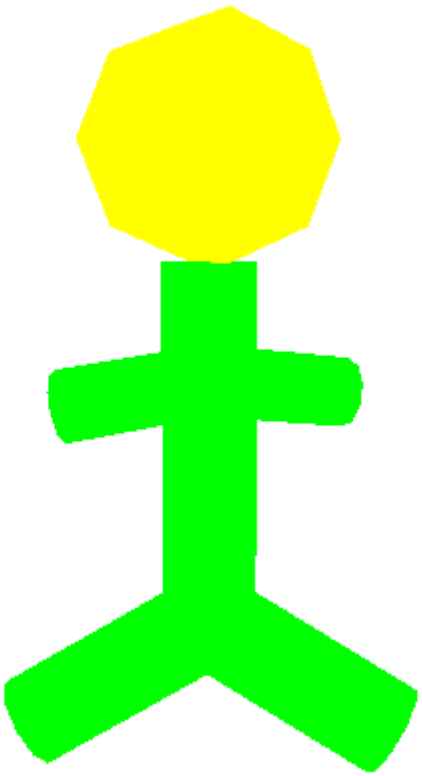
I Added the required animation, but I also added an animation for the arms to flail about while the stickman walks.



```
/*************************************************
*
* Example Seven
*
* The stickman program. This example shows
* how a hierarchical model can be constructed
* and displayed using OpenGL
*
*************************************************/

#ifdef WIN32
#include <Windows.h>
#endif
#include <GL/glew.h>
#define GLFW_DLL
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "Shaders.h"
#include <stdio.h>
#include <stack>

using namespace std;
```

```
GLuint program; // shader programs

glm::mat4 projection; // projection matrix
float eyex, eyey, eyez; // eye position
double theta, phi; // user's position on a sphere centered on the object
double r; // radius of the sphere

/*
* Structure for storing the information for a master
*/
struct Master {
GLuint vao;
int indices;
};

Master *body;
Master *head;
Master *leg;
Master *arm;
float walk =0.0;

/*
* The cylinder procedure constructs the master for cylinders.
* The parameters to this procedure are the radius of the
* cylinder, the height of the cylinder and the number of
* sides.
* The main axis of the cylinder is the z axis.
* The return value is a pointer to the new Master structure.
*/
Master *cylinder(double radius, double height, int sides) {
double *x; // the x coordinates of the vertices
double *y; // the y coordinates of the vertices
Master *result;
GLuint vao;
double angle;
double dangle;
int i;
GLfloat *vertices; // vertices to be stored in the buffer
GLushort *indices; // triangle indices to be stored in the buffer
int j;
int base;
GLuint vbuffer;
GLuint ibuffer;
GLint vPosition;

result = new Master;

glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
result->vao = vao;
result->indices = 3*4*sides;
```

```
/*
* Compute the x and y coordinates of the vertices
* around the edge of the sylinder
*/
x = new double[sides];
y = new double[sides];
dangle = 6.28/sides;
angle = 0.0;
for(i=0; i<sides; i++) {
x[i] = radius*cos(angle);
y[i] = radius*sin(angle);
angle += dangle;
}

vertices = new GLfloat[3*2*(sides+1)];
j = 0;

/* vertices on the bottom of the cylinder */
vertices[j++] = 0.0;
vertices[j++] = 0.0;
vertices[j++] = 0.0;
for(i=0; i<sides; i++) {
vertices[j++] = x[i];
vertices[j++] = y[i];
vertices[j++] = 0.0;
}

/* vertices on the top of the cylinder */
vertices[j++] = 0.0;
vertices[j++] = 0.0;
vertices[j++] = height;
for(i=0; i<sides; i++) {
vertices[j++] = x[i];
vertices[j++] = y[i];
vertices[j++] = height;
}

/*
* Now construct the triangle indices
*/
indices = new GLushort[3*4*sides];
j=0;

/* triangles of the bottom of the cylinder */
for(i=0; i<sides; i++) {
indices[j++] = 0;
indices[j++] = i+1;
indices[j++] = i+2;
}
indices[j-1] = 1;
```

```
/* triangles of the top of the cylinder */
base = sides+1;
for(i=0; i<sides; i++) {
indices[j++] = base;
indices[j++] = base+i+1;
indices[j++] = base+i+2;
}
indices[j-1] = base+1;

/* triangles on the sides of the cylinder */
for(i=1; i<sides; i++) {
indices[j++] = i;
indices[j++] = base+i;
indices[j++] = i+1;
indices[j++] = base+i;
indices[j++] = base+i+1;
indices[j++] = i+1;
}
indices[j++] = sides;
indices[j++] = base+sides;
indices[j++] = 1;
indices[j++] = base+sides;
indices[j++] = base+1;
indices[j++] = 1;

/*
* Now copy the vertices and indices into the appropriate buffers
*/
glGenBuffers(1, &vbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
glBufferData(GL_ARRAY_BUFFER, 3*2*(sides+1)*sizeof(GLfloat), vertices,
GL_STATIC_DRAW);
glGenBuffers(1, &ibuffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 3*4*sides*sizeof(GLushort), indices,
GL_STATIC_DRAW);

vPosition = glGetAttribLocation(program,"vPosition");
glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(vPosition);

return(result);
}

/*
* The init procedure first compiles and load the vertex
* and fragment shaders. It then creates the four masters
* that are used in the model.
*/
```

```
void init() {
int vs;
int fs;

/*
* compile and build the shader program
*/
vs = buildShader(GL_VERTEX_SHADER, (char*)"example7.vs");
fs = buildShader(GL_FRAGMENT_SHADER, (char*)"example7.fs");
program = buildProgram(vs,fs,0);
dumpProgram(program,(char*)"example 7");

glUseProgram(program);

body = cylinder(0.2,1.5,10);
leg = cylinder(0.2,0.9,10);
arm = cylinder(0.15,0.55,10);
head = cylinder(0.5,0.4,8);

}

/*
* Executed each time the window is resized,
* usually once at the start of the program.
*/
void framebufferSizeCallback(GLFWwindow *window, int w, int h) {

// Prevent a divide by zero, when window is too short
// (you cant make a window of zero width).

if (h == 0)
h = 1;

float ratio = 1.0f * w / h;

glfwMakeContextCurrent(window);

glViewport(0, 0, w, h);

projection = glm::perspective(0.7f, ratio, 1.0f, 100.0f);

}

/*
* This procedure is called each time the screen needs
* to be redisplayed
*/
void display() {
glm::mat4 model; // model matrix
glm::mat4 view; // view matrix
glm::mat4 viewPerspective; // combined view and perspective matrix
```

```
int viewLoc; // vertex shader location of viewPerspective matrix
int modelLoc; // shader location of model matrix
stack<glm::mat4> matrixStack; // stack of model matrices
int colourLoc; // colour location in fragment shader

/*
* Compute the viewing transformation and send it along
* with the perspective transformation to the vertex
* shader. Only need to do this once at the beginning
* of this procedure.
*/
view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
glm::vec3(0.0f, 0.0f, 1.0f),
glm::vec3(0.0f, 0.0f, 1.0f));
viewPerspective = projection * view;
model = glm::mat4(1.0);

glClear(GL_COLOR_BUFFER_BIT);
glUseProgram(program);
viewLoc = glGetUniformLocation(program,"viewPerspective");
glUniformMatrix4fv(viewLoc, 1, 0, glm::value_ptr(viewPerspective));
modelLoc = glGetUniformLocation(program,"model");
colourLoc = glGetUniformLocation(program,"colour");

/* Most of the body is green, send it to the fragment shader now */

glUniform4f(colourLoc, 0.0, 1.0, 0.0, 1.0);

/* draw the body */
glBindVertexArray(body->vao);
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glDrawElements(GL_TRIANGLES, body->indices, GL_UNSIGNED_SHORT, NULL);

/* draw right leg */
matrixStack.push(model);
model = glm::rotate(model, walk, glm::vec3(1.0, 0.0, 0.0));
model = glm::rotate(model, -2.3f, glm::vec3(0.0, 1.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glBindVertexArray(leg->vao);
glDrawElements(GL_TRIANGLES, leg->indices, GL_UNSIGNED_SHORT, NULL);
model = matrixStack.top();
matrixStack.pop();
/* draw left leg */
matrixStack.push(model);
model = glm::rotate(model, -walk, glm::vec3(1.0, 0.0, 0.0));
model = glm::rotate(model, 2.3f, glm::vec3(0.0, 1.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glBindVertexArray(leg->vao);
glDrawElements(GL_TRIANGLES, leg->indices, GL_UNSIGNED_SHORT, NULL);
model = matrixStack.top();
matrixStack.pop();
```

```
/* draw the right arm */
matrixStack.push(model);
model = glm::translate(model,glm::vec3(-0.1, 0.0, 1.0));
model = glm::rotate(model, -1.0f-walk, glm::vec3(0.0, 1.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glBindVertexArray(arm->vao);
glDrawElements(GL_TRIANGLES, arm->indices, GL_UNSIGNED_SHORT, NULL);
model = matrixStack.top();
matrixStack.pop();

/* draw right arm */
matrixStack.push(model);
model = glm::translate(model,glm::vec3(0.1, 0.0, 1.0));
model = glm::rotate(model, 1.0f+walk, glm::vec3(0.0, 1.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glBindVertexArray(arm->vao);
glDrawElements(GL_TRIANGLES, arm->indices, GL_UNSIGNED_SHORT, NULL);
model = matrixStack.top();
matrixStack.pop();

/* head colour - yellow */

glUniform4f(colourLoc, 1.0, 1.0, 0.0, 1.0);

/* draw head */
matrixStack.push(model);
model = glm::translate(model,glm::vec3(0.0, 0.2, 2.0));
model = glm::rotate(model, 1.57f, glm::vec3(1.0, 0.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
glBindVertexArray(head->vao);
glDrawElements(GL_TRIANGLES, head->indices, GL_UNSIGNED_SHORT, NULL);
model = matrixStack.top();
matrixStack.pop();

}

/*
* Called each time a key is pressed on
* the keyboard.
*/
static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
glfwSetWindowShouldClose(window, GLFW_TRUE);

if (key == GLFW_KEY_A && action == GLFW_PRESS)
phi -= 0.1;
if (key == GLFW_KEY_D && action == GLFW_PRESS)
phi += 0.1;
if (key == GLFW_KEY_W && action == GLFW_PRESS)
```

```
theta += 0.1;
if (key == GLFW_KEY_S && action == GLFW_PRESS)
theta -= 0.1;

eyex = (float)(r*sin(theta)*cos(phi));
eyey = (float)(r*sin(theta)*sin(phi));
eyez = (float)(r*cos(theta));

}

void error_callback(int error, const char* description)
{
fprintf(stderr, "Error: %s\n", description);
}


int main(int argc, char **argv) {
GLFWwindow *window;
float rate = 0.005;
float dwalk = 0.005;

// start by setting error callback in case something goes wrong

glfwSetErrorCallback(error_callback);

// initialize glfw

if (!glfwInit()) {
fprintf(stderr, "can't initialize GLFW\n");
}

// create the window used by our application

window = glfwCreateWindow(512, 512, "Example Seven", NULL, NULL);

if (!window)
{
glfwTerminate();
exit(EXIT_FAILURE);
}

// establish framebuffer size change and input callbacks

glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);

glfwSetKeyCallback(window, key_callback);

/*
* initialize glew
*/
glfwMakeContextCurrent(window);
```

```
GLenum error = glewInit();
if(error != GLEW_OK) {
printf("Error starting GLEW: %s\n",glewGetErrorString(error));
exit(0);
}

projection = glm::perspective(0.7f, 1.0f, 1.0f, 100.0f);

eyex = 0.0;
eyey = 6.0;
eyez = 0.0;

theta = 1.3;
phi = -1.5;
r = 6.0;

init();

glClearColor(1.0,1.0,1.0,1.0);

glfwSwapInterval(1);

// GLFW main loop, display model, swapbuffer and check for input

while (!glfwWindowShouldClose(window)) {
display();
glfwSwapBuffers(window);
glfwPollEvents();
walk += dwalk;
if (walk > 1.0)
dwalk = -rate;
if (walk < -1.0)
dwalk = rate;
}

glfwTerminate();

}
```