ANGRY BIRD SPACE

3D space cube physic game simulaton

Introduction of Our Project

Inspired by Rovio’s hit game “Angry bird”, we wanted to make a 3D version of the game, making our own physics engine from scratch in OpenGL.

DEMO VIDEO

We made two physics engines

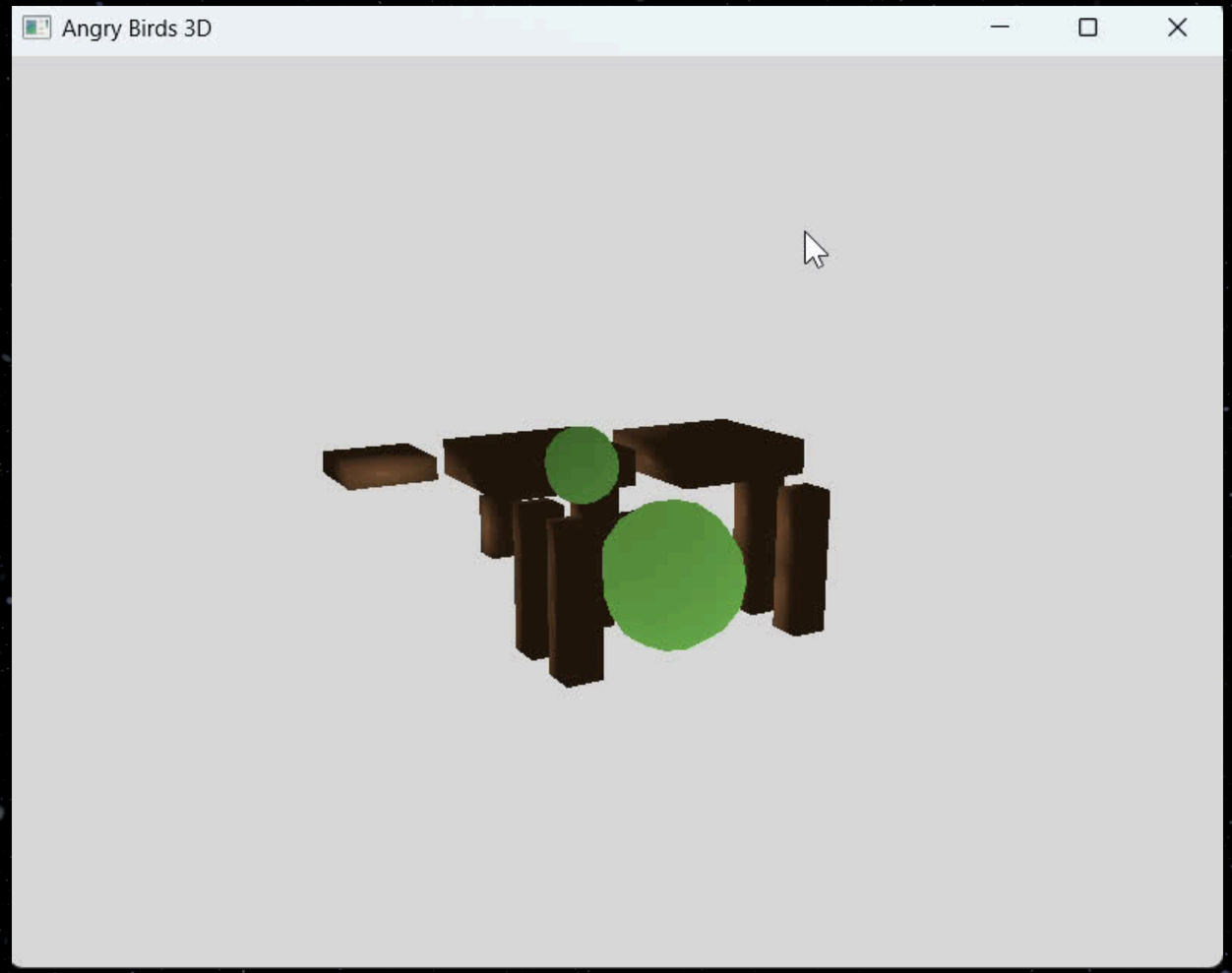
Gravity and rotational physics were big challenges in our project. We simplified the problem to work on these issues separately.

- One engine with no rotation, but supports gravity
  - Presented in progress report 2
- One engine with rotation physics, but no gravity
  - Submmited this engine as the final

Gravity Simulation

- For free falling objects: update the y-velocity by a set amount every tick.
  - Check if the object is resting on any object
- To prevent infinite bounce, set y-velocity to 0 when it reaches below a defined threshold.
- Difficult to implement with rotation physics, but we made it work with non-rotating objects
  - Jitter problem

Gravity Demo



First engine - Limitations

- No rotational physics
- Cuboids and Spheres only
- No textures

Final engine

GLAD

GLFW

GLM

IRRKLANG

DYNAMICS

COLLISION DETECTION

COLLISION RESPONSE

Rigid Body Dynamics

Note: the text of this page is from my original website, and I haven't updated it in a while. Make sure you look at the physics category for all of the articles related to rigid body dynamics.

Contents

Physics Articles
Physics Samples
Physics References

I started getting interested in high end physical simulation for games sometime in 1995. Since I didn't know anything about physics or know any real math (I had calculus in high school and enjoyed it, but had forgotten most of it), I had to teach myself everything from scratch. Let me just say it was a total blast. I *highly* recommend everyone taking the time to learn something really big and new every once in a while, in addition to all the little things we're all [hopefully] learning every day. As a bonus, it seems that the more we learn, the faster we're able to learn even newer things, which makes the whole process even more pleasurable.

As much fun as I had figuring this stuff out, I sure would have appreciated some references and articles aimed at my level when I was starting. So, I've created this web page in the hopes I can help everyone else get past those difficult first steps.

Physics Articles

I wrote a total of four articles about rigid body dynamics for Game Developer Magazine. I've posted them as PDF files, so they look just like they do in the magazine. There are downsides to PDF, however, including

IMPULSE BASED RIGID BODY PHYSICS

$$\mathbf{v}_2^A = \mathbf{v}_1^A + \frac{j}{M^A}\mathbf{n}$$

$$\omega_2^A = \omega_1^A + \frac{\mathbf{r}_\perp^{AP} \cdot j\mathbf{n}}{I^A}$$

$$j = \frac{-(1+e)\,\mathbf{v}_1^{AB} \cdot \mathbf{n}}{\mathbf{n}\cdot\mathbf{n}\left(\frac{1}{M^A} + \frac{1}{M^B}\right) + \frac{\left(\mathbf{r}_\perp^{AP} \cdot \mathbf{n}\right)^2}{I^A} + \frac{\left(\mathbf{r}_\perp^{BP} \cdot \mathbf{n}\right)^2}{I^B}}$$

MASS

RESTITUTION

INERTIA

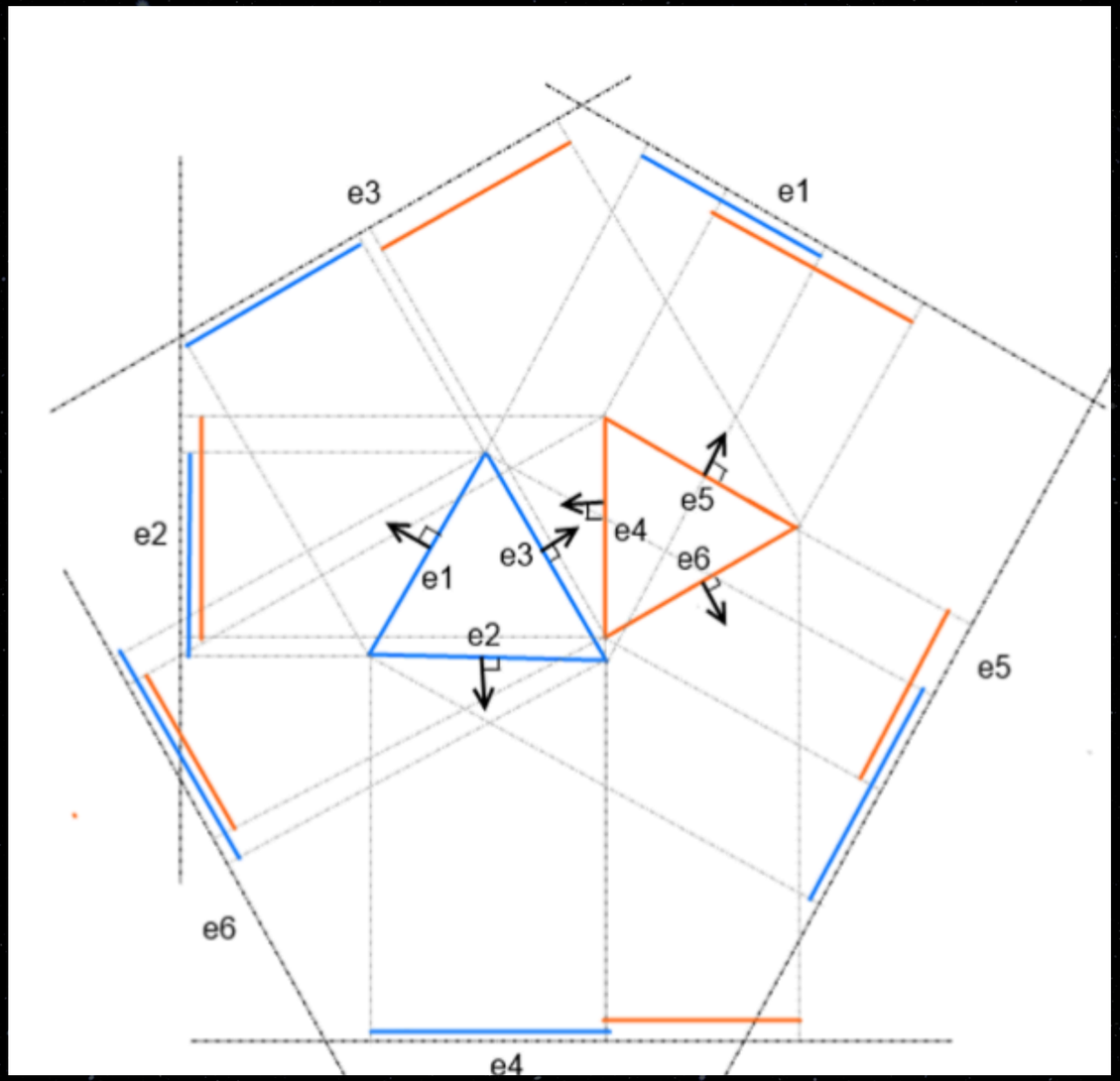
```
glm::mat3x3 inertia;
float mass;
float restitution;
```

$$I_z = \frac{1}{12} m\left(l^2 + w^2\right)$$

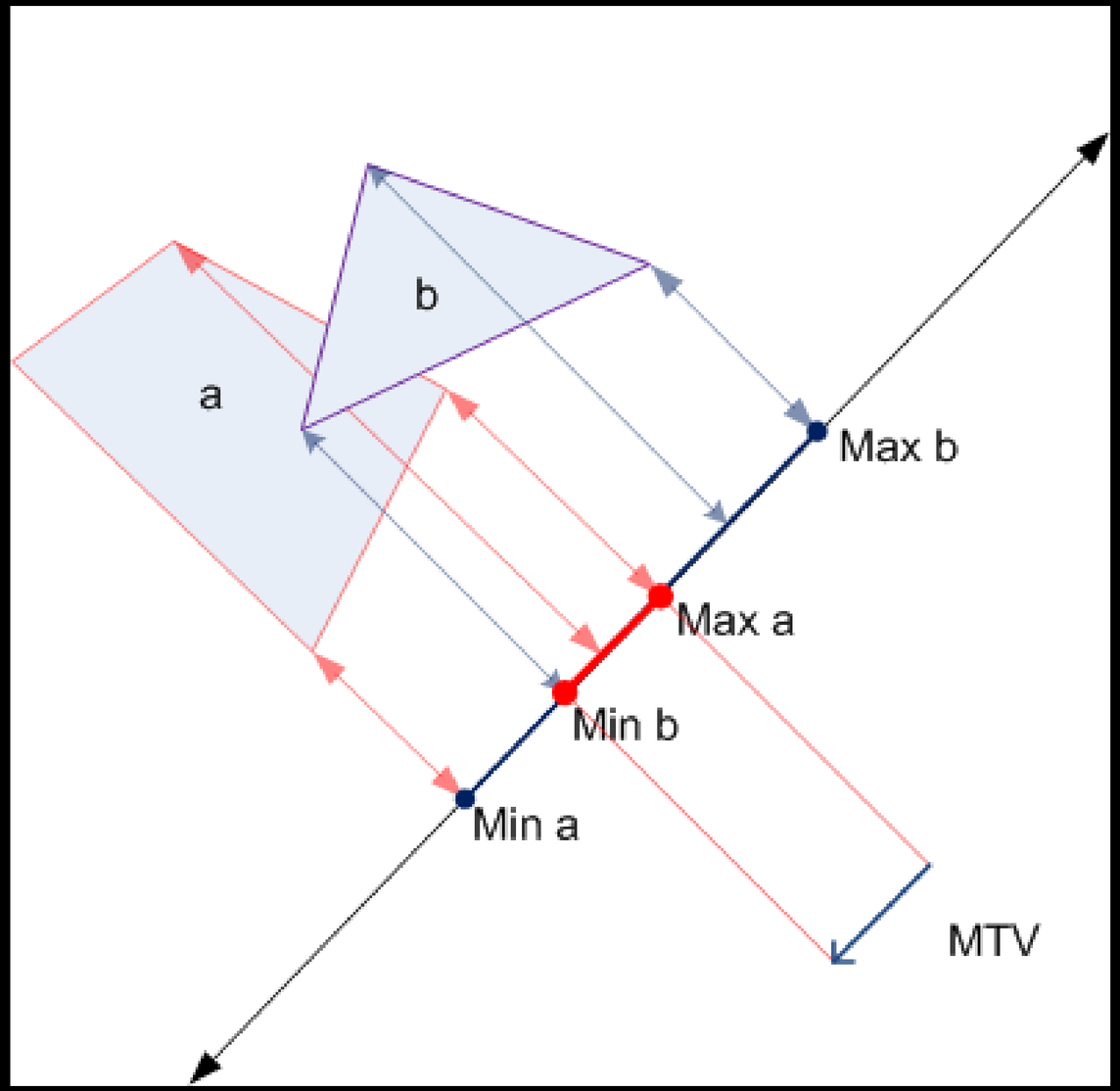
```
inertia = glm::mat3x3(1.0f);
float Ix = (mass / 12.0f) * (this->GetScale().y * this->GetScale().y + this->GetScale().z * this->GetScale().z);
float Iy = (mass / 12.0f) * (this->GetScale().x * this->GetScale().x + this->GetScale().z * this->GetScale().z);
float Iz = (mass / 12.0f) * (this->GetScale().x * this->GetScale().x + this->GetScale().y * this->GetScale().y);

inertia[0][0] = Ix;
inertia[1][1] = Iy;
inertia[2][2] = Iz;
```

SEPERATING AXIS THEOREM



MINIMUM TRANSLATION VECTOR



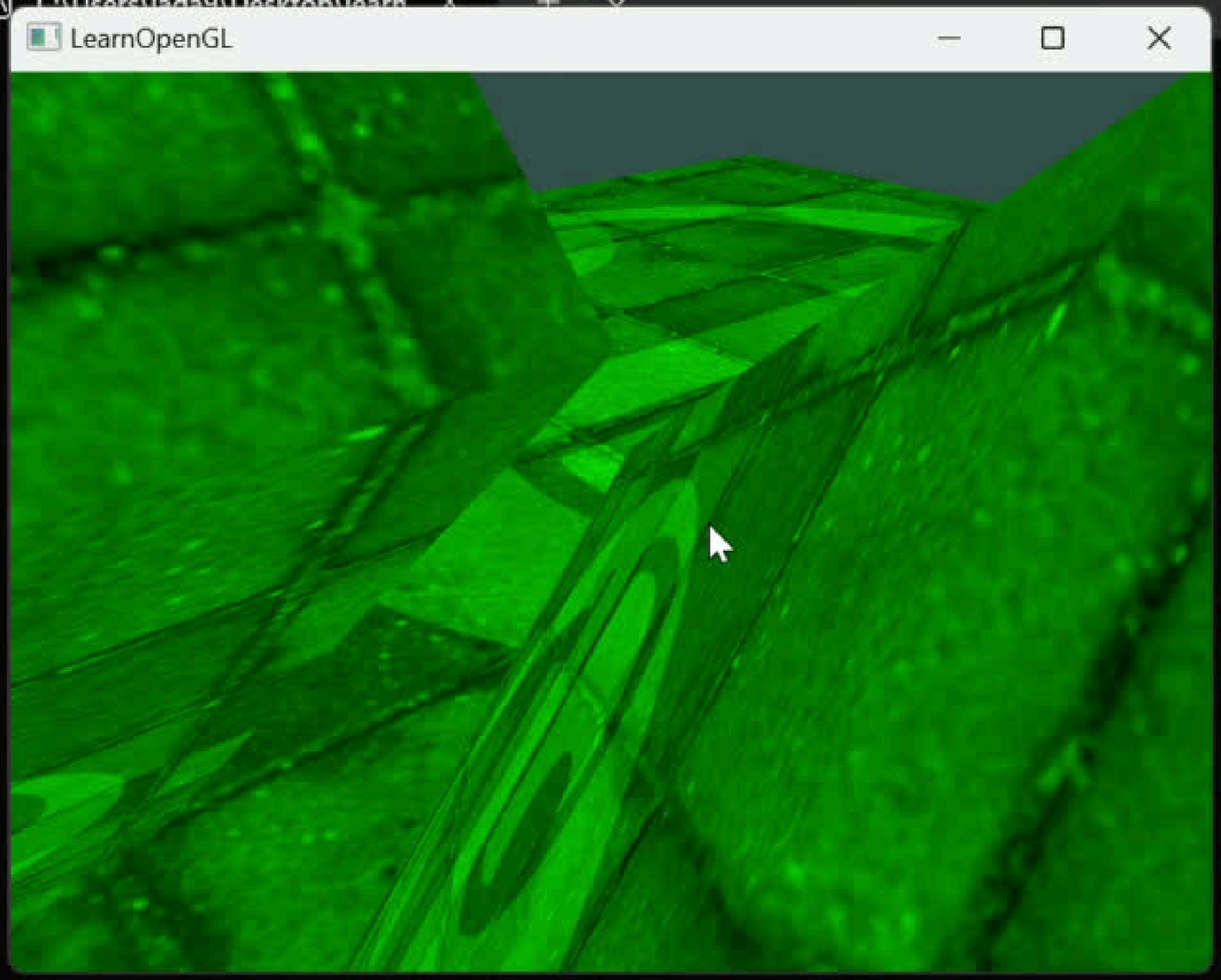
COLLISION TYPE

FACE-VERTEX

IF MINIMUM SA IS BASED ON FACE NORMAL

EDGE-EDGE

IF MINIMUM SA IS BASED ON CROSS EDGE


LearnOpenGL

COLLISION POINT

FACE-VERTEX

TAKE THE VERTEX POINT AS THE COLLISION POINT

EDGE-EDGE

TAKE THE AVERAGE OF THE MINIMUM SEPERATING DISTANCE BETWEEN
THE TWO EDGE AS THE COLLISION POINT

COLLISION RESPONSE

WE HAVE OUR DYNAMICS

WE HAVE OUR COLLISION POINTS

BUT MISSING ONE MORE THING

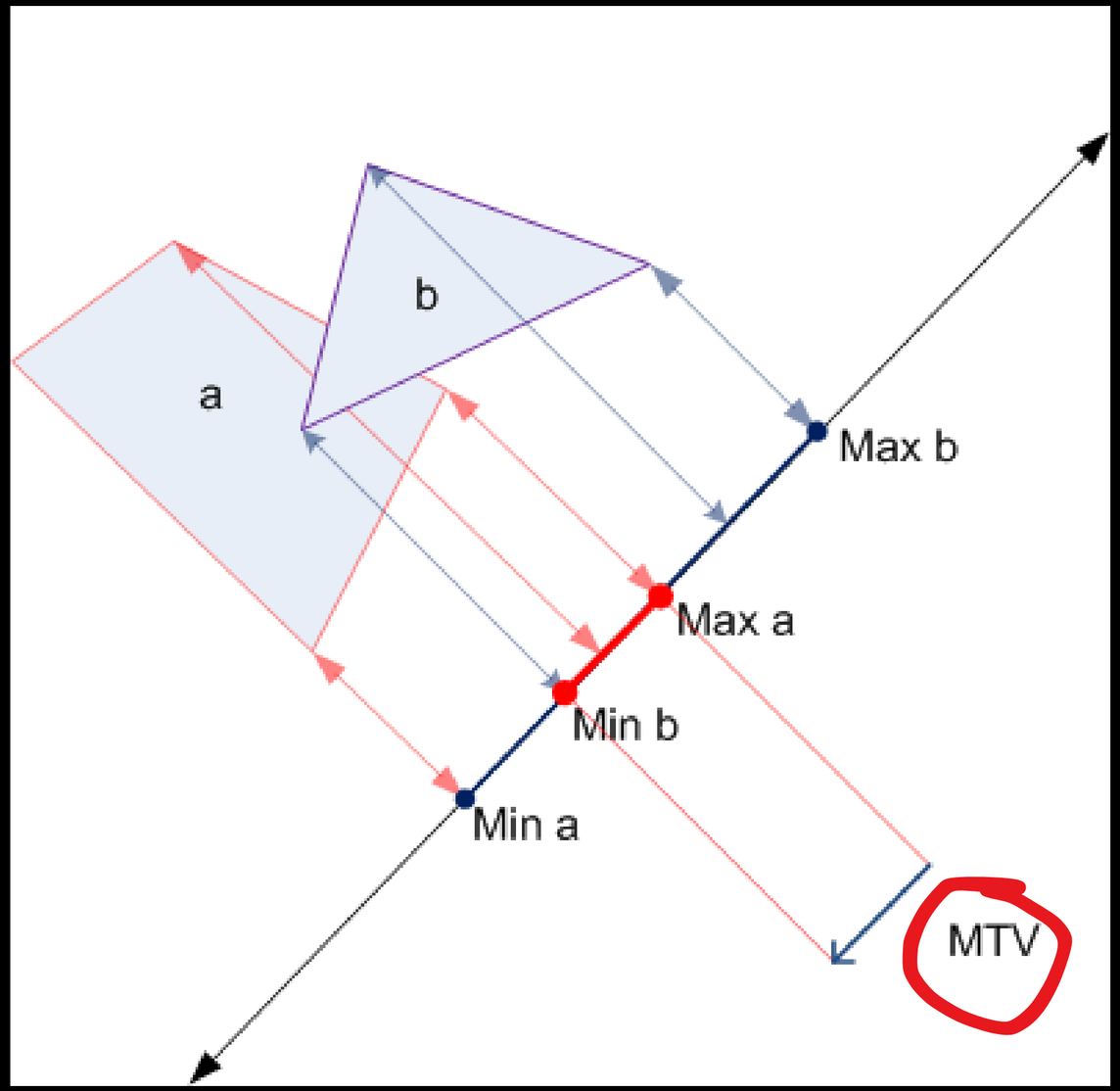
WHAT IS OUR COLLISION NORMAL?

$$\mathbf{v}_2^A = \mathbf{v}_1^A + \frac{j}{M^A}\mathbf{n}$$

$$\omega_2^A = \omega_1^A + \frac{\mathbf{r}_\perp^{AP} \cdot j\mathbf{n}}{I^A}$$

$$j = \frac{-(1+e)\,\mathbf{v}_1^{AB} \cdot \mathbf{n}}{\mathbf{n}\cdot\mathbf{n}\left(\frac{1}{M^A} + \frac{1}{M^B}\right) + \frac{\left(\mathbf{r}_\perp^{AP} \cdot \mathbf{n}\right)^2}{I^A} + \frac{\left(\mathbf{r}_\perp^{BP} \cdot \mathbf{n}\right)^2}{I^B}}$$

MINIMUM TRANSLATION VECTOR



SIMPLE!

USE MTV AS OUR COLLISION NORMAL

FIRST WORKING RIGID BODY PHYSIC IMPLEMENTATION

SHADERS

```
136     // build and compile our shader zprogram
137     // ------------------------------------
138     Shader ourShader("Linking\\shader\\shader.vs", "Linking\\shader\\shader.fs");
139     ourShader.use();
140
141     // load and create textures
142     // -----------------------
143     unsigned int texture1, texture2, texture3, texture4, texture5;
144     // texture 1
145     // ---------
146     glGenTextures(1, &texture1);
147     glBindTexture(GL_TEXTURE_2D, texture1);
148     // set the texture wrapping parameters
149     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);   // set texture wrapping to GL_REPEAT (default wrapp
150     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
151
152     // set texture filtering parameters
153     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
154     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
155     // load image, create texture and generate mipmaps
156     int width, height, nrChannels;
157     stbi_set_flip_vertically_on_load(true); // tell stb_image.h to flip loaded texture's on the y-axis.
158     unsigned char* data = stbi_load("Linking\\texture\\stone.png", &width, &height, &nrChannels, 0);
159     if (data)
160     {
161         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
162         glGenerateMipmap(GL_TEXTURE_2D);
163     }
164     else
165     {
166         std::cout << "Failed to load texture 1" << std::endl;
167     }
168     stbi_image_free(data);
169     // texture 2
```

SHADER CLASS

```
class Shader
{
public:
    unsigned int ID;
    // constructor generates the shader on the fly
    // ------------------------------------------------------------------------
    Shader(const char* vertexPath, const char* fragmentPath)
    {
        // 1. retrieve the vertex/fragment source code from filePath
        std::string vertexCode;
        std::string fragmentCode;
        std::ifstream vShaderFile;
        std::ifstream fShaderFile;
        // ensure ifstream objects can throw exceptions:
        vShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);
        fShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);
        try
        {
            // open files
            vShaderFile.open(vertexPath);
            fShaderFile.open(fragmentPath);
            std::stringstream vShaderStream, fShaderStream;
            // read file's buffer contents into streams
            vShaderStream << vShaderFile.rdbuf();
            fShaderStream << fShaderFile.rdbuf();
            // close file handlers
            vShaderFile.close();
            fShaderFile.close();
            // convert stream into string
            vertexCode = vShaderStream.str();
            fragmentCode = fShaderStream.str();
        }
        catch (std::ifstream::failure& e)
        {
            std::cout << "ERROR::SHADER::FILE_NOT_SUCCESSFULLY_READ: " << e.what() << std::endl;
        }
        const char* vShaderCode = vertexCode.c_str();
        const char* fShaderCode = fragmentCode.c_str();
        // 2. compile shaders
        unsigned int vertex, fragment;
        // vertex shader
        vertex = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vertex, 1, &vShaderCode, NULL);
        glCompileShader(vertex);
        checkCompileErrors(vertex, "VERTEX");
        // fragment Shader
        fragment = glCreateShader(GL_FRAGMENT_SHADER);
        glShaderSource(fragment, 1, &fShaderCode, NULL);
        glCompileShader(fragment);
        checkCompileErrors(fragment, "FRAGMENT");
        // shader Program
        ID = glCreateProgram();
        glAttachShader(ID, vertex);
        glAttachShader(ID, fragment);
        glLinkProgram(ID);
        checkCompileErrors(ID, "PROGRAM");
        // delete the shaders as they're linked into our program now and no longer necessary
        glDeleteShader(vertex);
        glDeleteShader(fragment);
```

FRAGMENT SHADER

```glsl
#version 330 core
out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

// texture samplers
uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;
uniform sampler2D texture4;
uniform sampler2D texture5;

// cube type
uniform int cubeType;

uniform vec3 color;
void main()
{
    vec4 texColor;
    if (cubeType == 1) {
        texColor = texture(texture1, TexCoord);
    } else if (cubeType == 2) {
        texColor = texture(texture2, TexCoord);
    } else if (cubeType == 3) {
        texColor = texture(texture3, TexCoord);
    } else if (cubeType == 4) {
        texColor = texture(texture4, TexCoord);
    } else if (cubeType == 5) {
        texColor = texture(texture5, TexCoord);
    }
    FragColor = vec4(color, 0.5) * texColor;
}
```

VERTEX SHADER

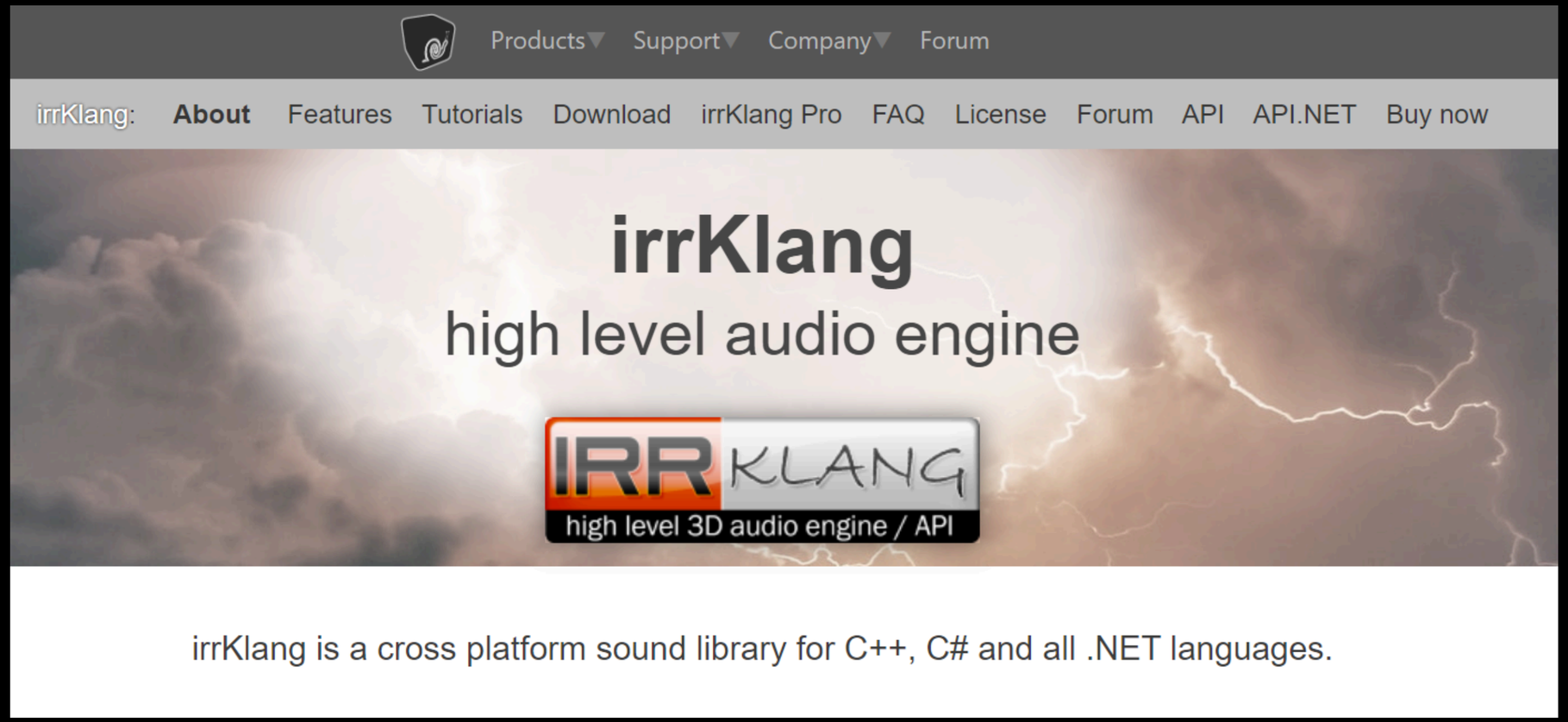
```glsl
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;


uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

SOUND


Products
Support
Company
Forum
irrKlang:
About
Features
Tutorials
Download
irrKlang Pro
FAQ
License
Forum
API
API.NET
Buy now
irrKlang
high level audio engine
IRRKLANG
high level 3D audio engine / API
irrKlang is a cross platform sound library for C++, C# and all .NET languages.

```
using namespace irrklang;

ISoundEngine* SoundEngine = createIrrKlangDevice();
```

```
SoundEngine->play2D("audio/bgm.mp3", true);
```