3D C/C++ tutorials - OpenGL 2.1 - Water waves GPU algorithm

To compile and run these tutorials some or all of these libraries are required: FreeImage 3.16.0, GLEW 1.11.0, GLUT 3.7.6 / GLUT for Dev-C++, GLM 0.9.5.4

Getting started in Visual Studio Express 2013

opengl_21_tutorials_win32_framework.h

```
...

#define WMR 128 // water mesh resolution
#define WHMR 128 // water height map resolution
#define WNMR 256 // water normal map resolution

class COpenGLRenderer
{
protected:
    int Width, Height;
    mat3x3 NormalMatrix;
    mat4x4 ModelMatrix, ViewMatrix, ViewMatrixInverse, ProjectionMatrix, ProjectionBiasMatrixInverse;

protected:
    CTexture PoolSkyCubeMap;
    GLuint WaterHeightMaps[2], WHMID, WaterNormalMap, PoolSkyVBO, WaterVBO, FBO;
    CShaderProgram WaterAddDropProgram, WaterHeightMapProgram, WaterNormalMapProgram, PoolSkyProgram, WaterProgram;
    int QuadsVerticesCount;

public:
    bool WireFrame, Pause;
    float DropRadius;

public:
    CString Text;

public:
    COpenGLRenderer();
    ~COpenGLRenderer();

    bool Init();
    void Render(float FrameTime);
    void Resize(int Width, int Height);
    void Destroy();

    void AddDrop(float x, float y, float DropRadius);
    void AddDropByMouseClick(int x, int y);
};

...
```

opengl_21_tutorials_win32_framework.cpp

```
...

COpenGLRenderer::COpenGLRenderer()
{
    WHMID = 0;

    WireFrame = false;
    Pause = false;

    DropRadius = 4.0f / 128.0f;

    Camera.SetViewMatrixPointer(&ViewMatrix, &ViewMatrixInverse);
}

COpenGLRenderer::~COpenGLRenderer()
{
}

bool COpenGLRenderer::Init()
{
    // ----------------------------------------------------------------------------------------------------------------------

    bool Error = false;

    // ----------------------------------------------------------------------------------------------------------------------

    if(!GLEW_ARB_texture_non_power_of_two)
    {
        ErrorLog.Append("GL_ARB_texture_non_power_of_two not supported!\r\n");
        Error = true;
    }

    if(!GLEW_ARB_texture_float)
    {
        ErrorLog.Append("GL_ARB_texture_float not supported!\r\n");
        Error = true;
    }

    if(!GLEW_EXT_framebuffer_object)
    {
        ErrorLog.Append("GL_EXT_framebuffer_object not supported!\r\n");
        Error = true;
    }
```

```
// ------------------------------------------------------------------------------------------------------------------------

char *PoolSkyCubeMapFileNames[] = {"pool\\right.jpg", "pool\\left.jpg", "pool\\bottom.jpg", "pool\\top.jpg", "pool\\front.jpg", "pool\\back.jpg"};

Error |= !PoolSkyCubeMap.LoadTextureCubeMap(PoolSkyCubeMapFileNames);

// ------------------------------------------------------------------------------------------------------------------------

Error |= !WaterAddDropProgram.Load("wateradddrop.vs", "wateradddrop.fs");
Error |= !WaterHeightMapProgram.Load("waterheightmap.vs", "waterheightmap.fs");
Error |= !WaterNormalMapProgram.Load("waternormalmap.vs", "waternormalmap.fs");
Error |= !PoolSkyProgram.Load("poolsky.vs", "poolsky.fs");
Error |= !WaterProgram.Load("water.vs", "water.fs");

// ------------------------------------------------------------------------------------------------------------------------

if(Error)
{
    return false;
}

// ------------------------------------------------------------------------------------------------------------------------

vec3 LightPosition = vec3(0.0f, 5.5f, -9.5f);

vec3 CubeMapNormals[6] = {
    vec3(-1.0f, 0.0f, 0.0f),
    vec3(1.0f, 0.0f, 0.0f),
    vec3(0.0f, -1.0f, 0.0f),
    vec3(0.0f, 1.0f, 0.0f),
    vec3(0.0f, 0.0f, -1.0f),
    vec3(0.0f, 0.0f, 1.0f),
};

// ------------------------------------------------------------------------------------------------------------------------

glUseProgram(WaterHeightMapProgram);
glUniform1f(glGetUniformLocation(WaterHeightMapProgram, "ODWHMR"), 1.0f / (float)WHMR);
glUseProgram(0);

glUseProgram(WaterNormalMapProgram);
glUniform1f(glGetUniformLocation(WaterNormalMapProgram, "ODWNMR"), 1.0f / (float)WNMR);
glUniform1f(glGetUniformLocation(WaterNormalMapProgram, "WMSDWNMRM2"), 2.0f / (float)WNMR * 2.0f);
glUseProgram(0);

glUseProgram(WaterProgram);
glUniform1i(glGetUniformLocation(WaterProgram, "WaterHeightMap"), 0);
glUniform1i(glGetUniformLocation(WaterProgram, "WaterNormalMap"), 1);
glUniform1i(glGetUniformLocation(WaterProgram, "PoolSkyCubeMap"), 2);
glUniform1f(glGetUniformLocation(WaterProgram, "ODWMS"), 1.0f / 2.0f);
glUniform3fv(glGetUniformLocation(WaterProgram, "LightPosition"), 1, &LightPosition);
glUniform3fv(glGetUniformLocation(WaterProgram, "CubeMapNormals"), 6, (float*)CubeMapNormals);
glUseProgram(0);

// ------------------------------------------------------------------------------------------------------------------------

WaterAddDropProgram.UniformLocations = new GLuint[2];
WaterAddDropProgram.UniformLocations[0] = glGetUniformLocation(WaterAddDropProgram, "DropRadius");
WaterAddDropProgram.UniformLocations[1] = glGetUniformLocation(WaterAddDropProgram, "Position");

WaterProgram.UniformLocations = new GLuint[1];
WaterProgram.UniformLocations[0] = glGetUniformLocation(WaterProgram, "CameraPosition");

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(2, WaterHeightMaps);

vec4 *Heights = new vec4[WHMR * WHMR];

for(int i = 0; i < WHMR * WHMR; i++)
{
    Heights[i] = vec4(0.0f, 0.0f, 0.0f, 0.0f);
}

for(int i = 0; i < 2; i++)
{
    glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[i]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, gl_max_texture_max_anisotropy_ext);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16F, WHMR, WHMR, 0, GL_RGBA, GL_FLOAT, Heights);
    glGenerateMipmapEXT(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, 0);
}

delete [] Heights;

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(1, &WaterNormalMap);

vec4 *Normals = new vec4[WNMR * WNMR];

for(int i = 0; i < WNMR * WNMR; i++)
{
    Normals[i] = vec4(0.0f, 1.0f, 0.0f, 1.0f);
}

glBindTexture(GL_TEXTURE_2D, WaterNormalMap);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, gl_max_texture_max_anisotropy_ext);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16F, WNMR, WNMR, 0, GL_RGBA, GL_FLOAT, Normals);
glGenerateMipmapEXT(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, 0);

delete [] Normals;
```

```
    // ------------------------------------------------------------------------------------------------------------------------

    glGenBuffers(1, &PoolSkyVBO);

    float PoolSkyVertices[] =
    {    // x, y, z, x, y, z, x, y, z, x, y, z
         1.0f, -1.0f, -1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f, -1.0f, // +X
        -1.0f, -1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,  1.0f, // -X
        -1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, // +Y
        -1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, // -Y
         1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f, // +Z
        -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f,  1.0f, -1.0f  // -Z
    };

    glBindBuffer(GL_ARRAY_BUFFER, PoolSkyVBO);
    glBufferData(GL_ARRAY_BUFFER, 288, PoolSkyVertices, GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    // ------------------------------------------------------------------------------------------------------------------------

    glGenBuffers(1, &WaterVBO);

    int WMRP1 = WMR + 1;

    vec3 *Vertices = new vec3[WMRP1 * WMRP1];

    float WMSDWMR = 2.0f / (float)WMR;

    for(int y = 0; y <= WMR; y++)
    {
        for(int x = 0; x <= WMR; x++)
        {
            Vertices[WMRP1 * y + x].x = x * WMSDWMR - 1.0f;
            Vertices[WMRP1 * y + x].y = 0.0f;
            Vertices[WMRP1 * y + x].z = 1.0f - y * WMSDWMR;
        }
    }

    CBuffer Quads;

    for(int y = 0; y < WMR; y++)
    {
        int yp1 = y + 1;

        for(int x = 0; x < WMR; x++)
        {
            int xp1 = x + 1;

            int a = WMRP1 * y + x;
            int b = WMRP1 * y + xp1;
            int c = WMRP1 * yp1 + xp1;
            int d = WMRP1 * yp1 + x;

            Quads.AddData(&Vertices[a], 12);
            Quads.AddData(&Vertices[b], 12);
            Quads.AddData(&Vertices[c], 12);
            Quads.AddData(&Vertices[d], 12);
        }
    }

    glBindBuffer(GL_ARRAY_BUFFER, WaterVBO);
    glBufferData(GL_ARRAY_BUFFER, Quads.GetDataSize(), Quads.GetData(), GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    QuadsVerticesCount = Quads.GetDataSize() / 12;

    Quads.Empty();

    delete [] Vertices;

    // ------------------------------------------------------------------------------------------------------------------------

    glGenFramebuffersEXT(1, &FBO);

    // ------------------------------------------------------------------------------------------------------------------------

    Camera.Look(vec3(0.0f, 1.0f, 2.5f), vec3(0.0f, -0.5f, 0.0f), true);

    // ------------------------------------------------------------------------------------------------------------------------

    srand(GetTickCount());

    // ------------------------------------------------------------------------------------------------------------------------

    return true;
}

void COpenGLRenderer::Render(float FrameTime)
{
    // add drops ----------------------------------------------------------------------------------------------------------

    if(!Pause)
    {
        static DWORD LastTime = GetTickCount();

        DWORD Time = GetTickCount();

        if(Time - LastTime > 100)
        {
            LastTime = Time;

            AddDrop(2.0f * (float)rand() / (float)RAND_MAX - 1.0f, 1.0f - 2.0f * (float)rand() / (float)RAND_MAX, 4.0f / 128.0f * (float)rand() / (float)R
        }
    }

    // update water surface ---------------------------------------------------------------------------------------------

    static DWORD LastTime = GetTickCount();

    DWORD Time = GetTickCount();

    if(Time - LastTime >= 16)
```

```
	{
		LastTime = Time;

		// update water height map --------------------------------------------------------------------------------------------

		glViewport(0, 0, WHMR, WHMR);

		GLuint whmid = (WHMID + 1) % 2;

		glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);
		glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, WaterHeightMaps[whmid], 0);
		glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_TEXTURE_2D, 0, 0);

		glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[WHMID]);
		glUseProgram(WaterHeightMapProgram);
		glBegin(GL_QUADS);
			glVertex2f(0.0f, 0.0f);
			glVertex2f(1.0f, 0.0f);
			glVertex2f(1.0f, 1.0f);
			glVertex2f(0.0f, 1.0f);
		glEnd();
		glUseProgram(0);
		glBindTexture(GL_TEXTURE_2D, 0);

		glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

		glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[whmid]);
		glGenerateMipmapEXT(GL_TEXTURE_2D);
		glBindTexture(GL_TEXTURE_2D, 0);

		++WHMID %= 2;

		// update water normal map --------------------------------------------------------------------------------------------

		glViewport(0, 0, WNMR, WNMR);

		glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);
		glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, WaterNormalMap, 0);
		glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_TEXTURE_2D, 0, 0);

		glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[WHMID]);
		glUseProgram(WaterNormalMapProgram);
		glBegin(GL_QUADS);
			glVertex2f(0.0f, 0.0f);
			glVertex2f(1.0f, 0.0f);
			glVertex2f(1.0f, 1.0f);
			glVertex2f(0.0f, 1.0f);
		glEnd();
		glUseProgram(0);
		glBindTexture(GL_TEXTURE_2D, 0);

		glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

		glBindTexture(GL_TEXTURE_2D, WaterNormalMap);
		glGenerateMipmapEXT(GL_TEXTURE_2D);
		glBindTexture(GL_TEXTURE_2D, 0);
	}

	// render pool sky mesh -----------------------------------------------------------------------------------------------

	glViewport(0, 0, Width, Height);

	glMatrixMode(GL_PROJECTION);
	glLoadMatrixf(&ProjectionMatrix);

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glEnable(GL_DEPTH_TEST);
	glEnable(GL_CULL_FACE);

	glMatrixMode(GL_MODELVIEW);
	glLoadMatrixf(&ViewMatrix);

	if(WireFrame)
	{
		glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	}

	glBindTexture(GL_TEXTURE_CUBE_MAP, PoolSkyCubeMap);
	glUseProgram(PoolSkyProgram);
	glBindBuffer(GL_ARRAY_BUFFER, PoolSkyVBO);
	glEnableClientState(GL_VERTEX_ARRAY);
	glVertexPointer(3, GL_FLOAT, 12, (void*)0);
	glDrawArrays(GL_QUADS, 0, 24);
	glDisableClientState(GL_VERTEX_ARRAY);
	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glUseProgram(0);
	glBindTexture(GL_TEXTURE_CUBE_MAP, 0);

	glDisable(GL_CULL_FACE);

	// render water surface -----------------------------------------------------------------------------------------------

	glActiveTexture(GL_TEXTURE0); glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[WHMID]);
	glActiveTexture(GL_TEXTURE1); glBindTexture(GL_TEXTURE_2D, WaterNormalMap);
	glActiveTexture(GL_TEXTURE2); glBindTexture(GL_TEXTURE_CUBE_MAP, PoolSkyCubeMap);
	glUseProgram(WaterProgram);
	glUniform3fv(WaterProgram.UniformLocations[0], 1, &Camera.Position);
	glBindBuffer(GL_ARRAY_BUFFER, WaterVBO);
	glEnableClientState(GL_VERTEX_ARRAY);
	glVertexPointer(3, GL_FLOAT, 12, (void*)0);
	glDrawArrays(GL_QUADS, 0, QuadsVerticesCount);
	glDisableClientState(GL_VERTEX_ARRAY);
	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glUseProgram(0);
	glActiveTexture(GL_TEXTURE2); glBindTexture(GL_TEXTURE_CUBE_MAP, 0);
	glActiveTexture(GL_TEXTURE1); glBindTexture(GL_TEXTURE_2D, 0);
	glActiveTexture(GL_TEXTURE0); glBindTexture(GL_TEXTURE_2D, 0);

	if(WireFrame)
	{
		glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

```
    }

    glDisable(GL_DEPTH_TEST);
}

void COpenGLRenderer::Resize(int Width, int Height)
{
    this->Width = Width;
    this->Height = Height;

    ProjectionMatrix = perspective(45.0f, (float)Width / (float)Height, 0.125f, 512.0f);
    ProjectionBiasMatrixInverse = inverse(ProjectionMatrix) * BiasMatrixInverse;
}

void COpenGLRenderer::Destroy()
{
    PoolSkyCubeMap.Destroy();

    WaterAddDropProgram.Destroy();
    WaterHeightMapProgram.Destroy();
    WaterNormalMapProgram.Destroy();
    PoolSkyProgram.Destroy();
    WaterProgram.Destroy();

    glDeleteTextures(2, WaterHeightMaps);
    glDeleteTextures(1, &WaterNormalMap);

    glDeleteBuffers(1, &PoolSkyVBO);
    glDeleteBuffers(1, &WaterVBO);

    if(GLEW_EXT_framebuffer_object)
    {
        glDeleteFramebuffersEXT(1, &FBO);
    }
}

void COpenGLRenderer::AddDrop(float x, float y, float DropRadius)
{
    if(x >= -1.0f && x <= 1.0f && y >= -1.0f && y <= 1.0f)
    {
        glViewport(0, 0, WMR, WMR);

        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);
        glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, WaterHeightMaps[(WHMID + 1) % 2], 0);
        glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_TEXTURE_2D, 0, 0);

        glBindTexture(GL_TEXTURE_2D, WaterHeightMaps[WHMID]);
        glUseProgram(WaterAddDropProgram);
        glUniform1f(WaterAddDropProgram.UniformLocations[0], DropRadius);
        glUniform2fv(WaterAddDropProgram.UniformLocations[1], 1, &vec2(x * 0.5f + 0.5f, 0.5f - y * 0.5f));
        glBegin(GL_QUADS);
            glVertex2f(0.0f, 0.0f);
            glVertex2f(1.0f, 0.0f);
            glVertex2f(1.0f, 1.0f);
            glVertex2f(0.0f, 1.0f);
        glEnd();
        glUseProgram(0);
        glBindTexture(GL_TEXTURE_2D, 0);

        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

        ++WHMID %= 2;
    }
}

void COpenGLRenderer::AddDropByMouseClick(int x, int y)
{
    float s = (float)x / (float)(Width - 1);
    float t = 1.0f - (float)y / (float)(Height - 1);

    vec4 Position = ViewMatrixInverse * (ProjectionBiasMatrixInverse * vec4(s, t, 0.5f, 1.0f));
    Position /= Position.w;

    vec3 Ray = normalize(*(vec3*)&Position - Camera.Position);

    vec3 Normal = vec3(0.0f, 1.0f, 0.0f);
    float D = -dot(Normal, vec3(0.0f, 0.0f, 0.0f));

    float NdotR = -dot(Normal, Ray);

    if(NdotR != 0.0f)
    {
        float Distance = (dot(Normal, Camera.Position) + D) / NdotR;

        if(Distance > 0.0f)
        {
            vec3 Position = Ray * Distance + Camera.Position;

            AddDrop(Position.x, Position.z, DropRadius);
        }
    }
}

...

void COpenGLView::OnKeyDown(UINT Key)
{
    switch(Key)
    {
        case VK_F1:
            OpenGLRenderer.WireFrame = !OpenGLRenderer.WireFrame;
            break;

        case '1':
            OpenGLRenderer.DropRadius = 4.0f / 256.0f;
            break;

        case '2':
            OpenGLRenderer.DropRadius = 4.0f / 128.0f;
            break;

        case '3':
            OpenGLRenderer.DropRadius = 4.0f / 64.0f;
            break;

        case '4':
            OpenGLRenderer.DropRadius = 4.0f / 32.0f;
            break;

        case '5':
            OpenGLRenderer.DropRadius = 4.0f / 16.0f;
            break;

        case VK_SPACE:
            OpenGLRenderer.Pause = !OpenGLRenderer.Pause;
            break;
    }
}

void COpenGLView::OnLButtonDown(int X, int Y)
{
    OpenGLRenderer.AddDropByMouseClick(X, Y);
}

...
```

wateradddrop.vs

```
#version 120

void main()
{
    gl_TexCoord[0] = gl_Vertex;
    gl_Position = gl_Vertex * 2.0 - 1.0;
}
```

wateradddrop.fs

```
#version 120

uniform sampler2D WaterHeightMap;

uniform float DropRadius;
uniform vec2 Position;

void main()
{
    vec2 vh = texture2D(WaterHeightMap, gl_TexCoord[0].st).rg;

    float d = distance(gl_TexCoord[0].st, Position);

    gl_FragColor = vec4(vh.r, vh.g - 4.0f * max(DropRadius - d, 0.0), 0.0, 0.0);
}
```

waterheightmap.vs

```
#version 120

void main()
{
    gl_TexCoord[0] = gl_Vertex;
    gl_Position = gl_Vertex * 2.0 - 1.0;
}
```

waterheightmap.fs

```
#version 120

uniform sampler2D WaterHeightMap;

uniform float ODWHMR;

void main()
{
    vec2 vh = texture2D(WaterHeightMap, gl_TexCoord[0].st).rg;

    float force = 0.0;

    force += 0.707107 * (texture2D(WaterHeightMap, gl_TexCoord[0].st - vec2(ODWHMR, ODWHMR)).g - vh.g);
    force += texture2D(WaterHeightMap, gl_TexCoord[0].st - vec2(0.0, ODWHMR)).g - vh.g;
    force += 0.707107 * (texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(ODWHMR, -ODWHMR)).g - vh.g);

    force += texture2D(WaterHeightMap, gl_TexCoord[0].st - vec2(ODWHMR, 0.0)).g - vh.g;
    force += texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(ODWHMR, 0.0)).g - vh.g;

    force += 0.707107 * (texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(-ODWHMR, ODWHMR)).g - vh.g);
    force += texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(0.0, ODWHMR)).g - vh.g;
    force += 0.707107 * (texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(ODWHMR, ODWHMR)).g - vh.g);

    force *= 0.125;

    vh.r += force;
    vh.g += vh.r;
    vh.g *= 0.99;

    gl_FragColor = vec4(vh, 0.0, 0.0);
}
```

waternormalmap.vs

```
#version 120

void main()
{
    gl_TexCoord[0] = gl_Vertex;
    gl_Position = gl_Vertex * 2.0 - 1.0;
}
```

waternormalmap.fs

```
#version 120

uniform sampler2D WaterHeightMap;

uniform float ODWNMR, WMSDWNMRM2;

void main()
{
    float y[4];

    y[0] = texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(ODWNMR, 0.0)).g;
    y[1] = texture2D(WaterHeightMap, gl_TexCoord[0].st + vec2(0.0, ODWNMR)).g;
    y[2] = texture2D(WaterHeightMap, gl_TexCoord[0].st - vec2(ODWNMR, 0.0)).g;
    y[3] = texture2D(WaterHeightMap, gl_TexCoord[0].st - vec2(0.0, ODWNMR)).g;

    vec3 Normal = normalize(vec3(y[2] - y[0], WMSDWNMRM2, y[1] - y[3]));

    gl_FragColor = vec4(Normal, 1.0);
}
```

poolsky.vs

```
#version 120

void main()
{
    gl_TexCoord[0].stp = vec3(gl_Vertex.x, -gl_Vertex.yz);
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

poolsky.fs

```
#version 120

uniform samplerCube PoolSkyCubeMap;

void main()
{
    gl_FragColor = textureCube(PoolSkyCubeMap, gl_TexCoord[0].stp);
}
```

water.vs

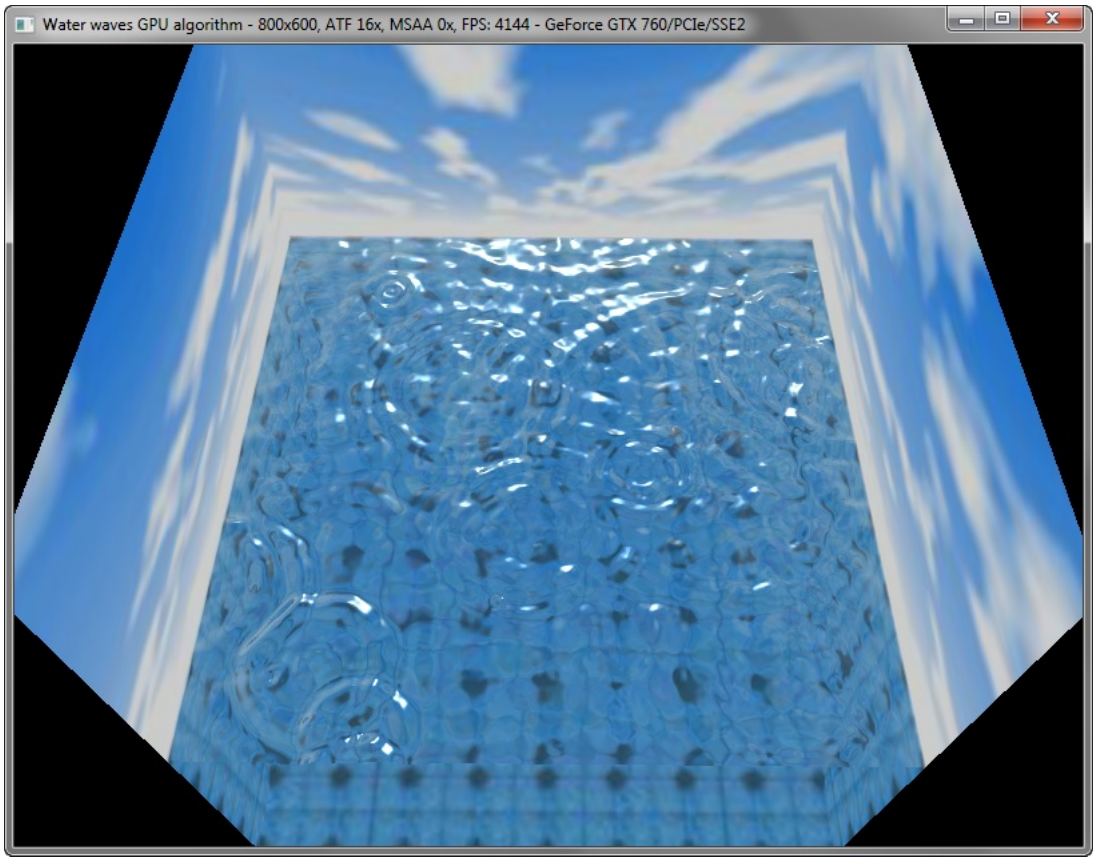
```
#version 120

uniform sampler2D WaterHeightMap;

varying vec3 Position;

void main()
{
    gl_TexCoord[0].st = vec2(gl_Vertex.x * 0.5 + 0.5, 0.5 - gl_Vertex.z * 0.5);
    Position = gl_Vertex.xyz;
    Position.y += texture2D(WaterHeightMap, gl_TexCoord[0].st).g;
    gl_Position = gl_ModelViewProjectionMatrix * vec4(Position, 1.0);
}
```

water.fs

```
#version 120

uniform sampler2D WaterNormalMap;
uniform samplerCube PoolSkyCubeMap;

uniform vec3 LightPosition, CubeMapNormals[6], CameraPosition;

varying vec3 Position;

vec3 IntersectCubeMap(vec3 Position, vec3 Direction)
{
    vec3 Point;

    for(int i = 0; i < 6; i++)
    {
        float NdotR = -dot(CubeMapNormals[i], Direction);

        if(NdotR > 0.0)
        {
            float Distance = (dot(CubeMapNormals[i], Position) + 1.0) / NdotR;

            if(Distance > -0.03)
            {
                Point = Direction * Distance + Position;

                if(Point.x > -1.001 && Point.x < 1.001 && Point.y > -1.001 && Point.y < 1.001 && Point.z > -1.001 && Point.z < 1.001)
                {
                    break;
                }
            }
        }
    }

    return vec3(Point.x, -Point.yz);
}

void main()
{
    vec3 Normal = normalize(texture2D(WaterNormalMap, gl_TexCoord[0].st).rgb);
    vec3 Direction = normalize(Position - CameraPosition);

    if(CameraPosition.y > 0)
    {
        vec3 ReflectedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, reflect(Direction, Normal))).rgb;
        vec3 RefractedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, refract(Direction, Normal, 0.750395))).rgb;

        vec3 LightDirectionReflected = reflect(normalize(Position -  LightPosition), Normal);

        float Specular = pow(max(-dot(Direction, LightDirectionReflected), 0.0), 128);

        gl_FragColor.rgb = mix(ReflectedColor, RefractedColor, -dot(Normal, Direction)) + Specular;
    }
    else
    {
        Normal = -Normal;

        vec3 ReflectedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, reflect(Direction, Normal))).rgb;
        vec3 DirectionRefracted = refract(Direction, Normal, 1.332631);

        if(DirectionRefracted.x == 0.0 && DirectionRefracted.y == 0.0 && DirectionRefracted.z == 0.0)
        {
            gl_FragColor.rgb = ReflectedColor;
        }
        else
        {
            vec3 RefractedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, DirectionRefracted)).rgb;
            gl_FragColor.rgb = mix(ReflectedColor, RefractedColor, -dot(Normal, Direction));
        }
    }
}
```

Download

water_waves_gpu_algorithm.zip (Visual Studio 2005 Professional)


Water waves GPU algorithm - 800x600, ATF 16x, MSAA 0x, FPS: 4144 - GeForce GTX 760/PCIe/SSE2

Comments (6)

zxx43, December 19, 2014, 04:27 PM

In waternormalmap.fs ODWNMR is 1.0 / 256.0 and WMSDWNMRM2 is 4.0 / 256.0, why WMSDWNMRM2 is not ODWNMR * 2.0?

Admin, December 19, 2014, 04:57 PM

ODWNMR is TEXTURE SPACE distance between 2 neighboring pixels (or a pixel size) of the water normal map texture and is used to fetch heights from the water height map texture - this is a trick - it also fetches heights between pixels of the height map texture, because the resolution of the normal map is 2x higher. WMSDWNMRM2 is WORLD SPACE doubled distance between two neghboring pixels (or a pixel size) of the water normal map texture and is used to calculate normals of the water mesh. Heights and normals are stored in the water height and normal map textures in WORLD SPACE. WORLD SPACE size of the water mesh is 2.0 x 2.0. TEXTURE SPACE size of any texture is 1.0 x 1.0.

zxx43, December 22, 2014, 09:52 AM

Why do we need to check in water.fs if the refract function returns vec3(0.0) only when the camera is under water?

Admin, December 22, 2014, 10:15 AM

http://en.wikipedia.org/wiki/Refraction
http://en.wikipedia.org/wiki/Refraction#mediaviewer/File:RefractionReflextion.svg

https://www.opengl.org/registry/doc/glspec21.20061201.pdf
https://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

The refract function is described in the chapter 8.4 of the OpenGL shading language 1.20.8.

```
genType refract(genType I, genType N, float eta)
{
    k = 1.0 - eta * eta * (1.0 - dot(N, I) * dot(N, I));

    if(k < 0.0)
    {
         return genType(0.0);
    }
    else
    {
        return eta * I - (eta * dot(N, I) + sqrt(k)) * N;
    }
}
```

In case of total internal reflection (k < 0.0) the refract function returns genType(0.0). The condition k < 0.0 can be true only when eta > 1.0, only when the camera is under water.

zxx43, December 26, 2014, 11:04 AM

How is it possible that waves are spreading, which equations did you use and which informations are stored in the components of the water height map texture?

Admin, December 26, 2014, 12:03 PM

Everything is in the code - in the "update water height map" part of the COpenGLRenderer::Render function and in the waterheightmap.fs fragment shader. You have to be able to understand it on your own - analyze it. Don't look for wave equations in this tutorial, we don't use and we don't need any. To create the effect of spreading waves we just use couple of very simple rules, which might unveil the deeper truth about waves. Very interesting consequences of those simple rules are wave magnitude and length reduction, interference of waves, reflection of waves at the edges of the water mesh and spreading of waves. The vertical force applied to a vertex of the water mesh is stored in the red component of the water height map texture, the actual height of a vertex of the water mesh is stored in the green component of the water height map texture and the blue and alpha components of the water height map textures are not used. The vertical force applied to a vertex of the water mesh is calculated as sum of weighted averages of the height differences of a vertex and it's surrounding vertices, this force changes the height and the height is then slightly reduced to attenuate waves by time - these are those simple rules that do the trick and they're obvious.

Add comment

Name

Password (to delete your comment)

Text

HTML tags are not allowed. Put code or formatted text between [CODE][/CODE] tags.

Add comment