Технически Университет – София

# КУРСОВ ПРОЕКТ

по дисциплина

**„Компютърна графика“**

на тема:

**Tessellation Shaders**

**Изготвил:**
Дияна Антовска, ф.No123216012

4 курс, 9 поток, 47 група

ФКСТ, КСИ, ПС

**Проверил:**
ас. Десислав Андреев

София 2019

# Въведение

[1]**Shader** в компютърната графика представлява вид копмютърна програма, която предимно се ползва за **shading** на 3D сцени - създаването на съответни нива на светлина, тъмнина и цветове в едно рендерирано изображение, т.е. определяне на цвета на точки чрез интерполиране между точките с известна осветеност. За тази цел се извършват различни спциализирани функции, които са част от специални ефекти на компютърната графика, пост-обработка на видеа несвързани със създаването на сенки или функции, които изобщо не са свързани с графиката.

Традиционните shader-и изчисляват ефектите на рендерите на графичния хардуер с висока степен на гъвкавост. Повечето shader-и са писани за и работят на GPU-то. С развитието на видео картите, голям брой библиотеки като OpenGL, Direct3D започнаха да поддържат shader-и. Съществуват 3 вида shader-и: 2D, 3D и Compute Shaders.

**2D shader-ите** се прилагат върху изображения, т.н. текстури в компютърната графика. Те модифицират атрибути на пикселите. Също могат да участват в рендирането на 3D геометрията. Единствения 2D shader е **Pixel Shader**-а. Pixel Shader-ите изчисляват цвета и други атрибути на всеки фрагмент – рендерираща единица, която влияе на един пиксел.

**3D shader-ите** се прилагат на 3D модели или геометрични обекти, но могат да достъпват цветовете и текстурите, които се ползват за изчертаване на модела или **mesh**-а (колекция от възли, ръбове и повърхнини, които дефинират обекта). **Vertex Shader** – най-често срещаните shader-и, целта на които е да трансформират 3D позицията на всеки възел в 2D координати във виртуалното пространство на екрана (също и стойността на дълбочината за z-buffer-а). Те влияят върху позиция, цвят и координати на текстурите, но не могат да създават нови възли. **Geometry Shader** – генерира примитиви като точки, линии и триъгълници при подадени друг вид примитиви. **Primitive Shader-ите** са близки с **Compute Shader-ите** (добавят допълнителни ефекти в алгоритмите за анимация и осветление), само че имат достъп до необходими данни за процесиране на геометрията. **Tessellation Shaders** – последния вид 3D shader-и и тяхната същност, функции и приложения са концентрацията в този реферат.

## Tessellation Shader – Същност, функции

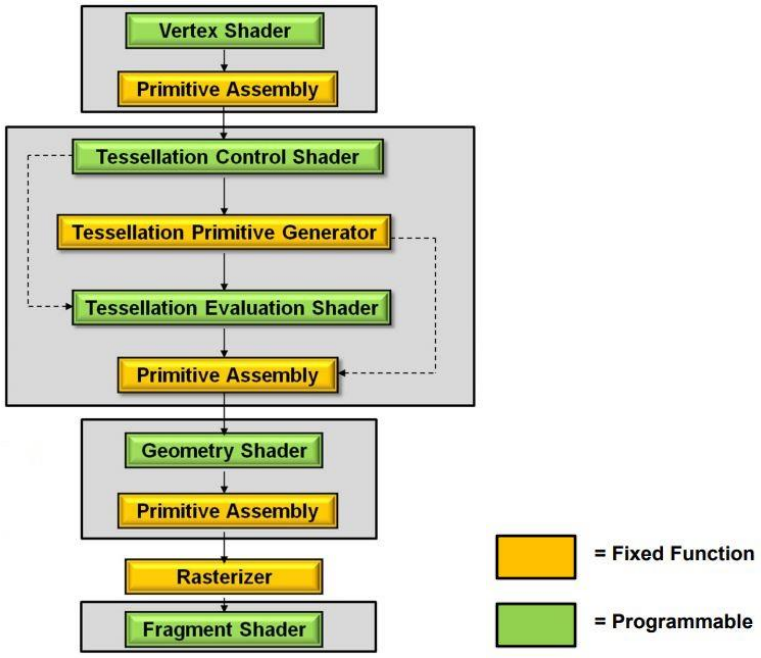
[2]Tessellation е процеса, който разделя една повърхност на гладки триъгълни mesh-ове. Tessellation Shader-ите интерполират геометрия (точки, линии, полигони) с цел създаване на допълнителна геометрия, която ще направи адаптивно подразделение в зависимост от размер или кривини (curves). В общи линии, тяхната цел е да увеличат качеството с изобразяване на релефа на геометричните модели в изходното изображение.

Geometry Shader-ите също имат способността за генериране на геометрия от вече съществуваща. Разликата е, че те могат да създадат нова, различна топология, докато Tessellation Shader-ите добавят такива елементи, които са същите като началните. Например, от един сегмент могат да се получат повече линии или от един триъгълник - повече триъгълници.

[3]Tessellation Shader-ите се използват за следното:

- Адаптивно подразделение в зависимост от различни критерии (размер, криви и т.н.)
- Могат да се предоставят по-груби модели, но да се покажат по-фини – geometric compression
- Може да се адаптира визуално качество до необходимото ниво на детайлност (level of detail - LOD)
- Могат да се представят по-гладки силуети

Tessellation Shader-а се прилага върху изображението след Vertex Shader-а и преди Geometry Shader-а.



Tessellation Shader-а се съдържа от Tessellation Control Shader (TCS), Tessellation Primitive (Pattern) Generator (TPG) и Tessellation Evaluation Shader (TES). TCS и TES са програмируеми, докато TPG е фиксирана функция. За да се активира Tessellation Shader-а задължително е преди това Vertex Shader-а да се изпълни за всеки възел, който е посочен от командата за рендериране.

Tessellation Shader-ите се появяват с OpenGL 4.0 и Direct3D 11. За тяхното програмиране се използва езика [7]GLSL – OpenGL Shading Language, които е shading език от високо ниво базиран на програмния език C. Създаден е от OpenGL Architecture Review Board (OpenGL ARB) с цел да се позволи на програмистите директно да достъпват и да контролират графичния пайплайн, а не да използват ARB Assembly Language или някой друг език от ниско ниво.

**Tessellation Control Shader (TCS)** – чете входа, който Vertex Shader-а му подава, който вход всъщност представлява **patch** (примитива, която се състои от n на брой възли). [4]Основната функция, която трябва да се извърши на това ниво е да се определи нивото на tessellation, което трябва да се приложи на дадена примитива и да се осъществи необходимата трансформация на входните данни. Тези изчисления се правят в зависимост от разстоянието до окото, обхвата на екрана, кривите на обектите, грапавостта на изместване.

TCS може да промени големината на примитивата, т.е. да добави повече или да премахне някои от възлите, но не може да отстрани самата примитива или да създаде нова. TCS е опционален, т.е. ако няма нужда от никаква трансформация, Vertex Shader-а ще изпрати данните директно на TPG.

[5]Има едно извикване за един изходен възел. Например, ако командата за рендериране изобрази 20 примитиви и всяка изходна примитива съдържа четири възела, TCS ще се извика 80 пъти – 80 x, y, z координати за всеки възел.

layout(vertices = patch_size) out;

patch_size – броя възли на изхода, обаче също определя броя извиквания на TCS.

**Вход**

in vec2 texCoord[];

Входовете от Vertex Shader-а към TCS са агрегирани в масиви, които зависят от големината на входния patch.

**Вградените** входни променливи в TCS са:

in int gl_PatchVerticesIn – броя възли във входния patch

in int gl_PrimitiveID; - индекса на текущия patch в рамките на рендериращата команда

in int gl_InvocationID; - индекса на TCS извикването в рамките на patch-а

in gl_PerVertex{

    vec4 gl_Position;

    float gl_PointSize;

    float gl_ClipDistance[];

}gl_in[gl_MaxPatchVertices]; - изхода от Vertex Shader-а

**Изход**

out vec2 vertexTexCoord[]; - големината винаги ще е колкото тази на изходния patch

**Вградени** изходни променливи

patch out float gl_TessLevelOuter[4];

patch out float gl_TessLevelInner[2];

gl_TessLevelOuter и gl_TessLevelInner дефинират нивата на tessellation използвани от TPG.

out gl_PerVertex{

    vec4 gl_Position;

    float gl_PointSize;

    float gl_ClipDistance[];

}gl_out[]; - опционални за всеки възел променливи

**Ограничения**

Максималната големина на изходни patches се дефинира от GL_MAX_PATCH_VERTICES. Минималната е 32.

Максималната стойност на компоненти за активни по възел изходни променливи е GL_MAX_TESS_CONTROL_OUTPUT_COMPONENTS. Минималната е 128.

Максималната стойност на компоненти за активни по patch изходни променливи е GL_MAX_TESS_PATCH_COMPONENTS. Минималната е 120.

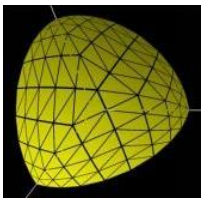
Съществува и ограничение за броя на компонентите, които могат да се включат в изходния patch. Стойноста се получава при умножаване на броя активни по възел изходни компоненти и броя на изходни възли, и се добави броя на активни по patch компоненти. Максималната стойност е GL_MAX_TESS_CONTROL_TOTAL_OUTPUT_COMPONENTS. Минималната е 4096.

**Tessellation Primitive Generator (TPG)** – Извиква се веднъж за всяка примитива – теселира кривата или повърхността в **u, v, w** координати само, ако TES е активен в текущата програма. Това е фиксирана функция. Като вход приема възлите от TCS и създава нови възли за триъгълниците, четириъгълници или изолиниите. За триъгълниците са необходими u, v и w, докато за останалите две - u и v.

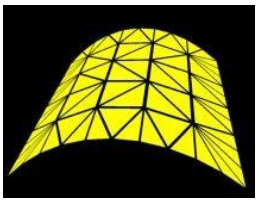
На TPG влияят следните фактори: нивата на tessellation, които зависят от TCS или тези по подразбиране, разстоянието между възлите, входните примитиви и реда, по който ще се генерират.

Tessellation Shader-ите могат да произведат вградените шаблони:

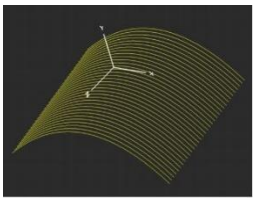
Triangle pattern

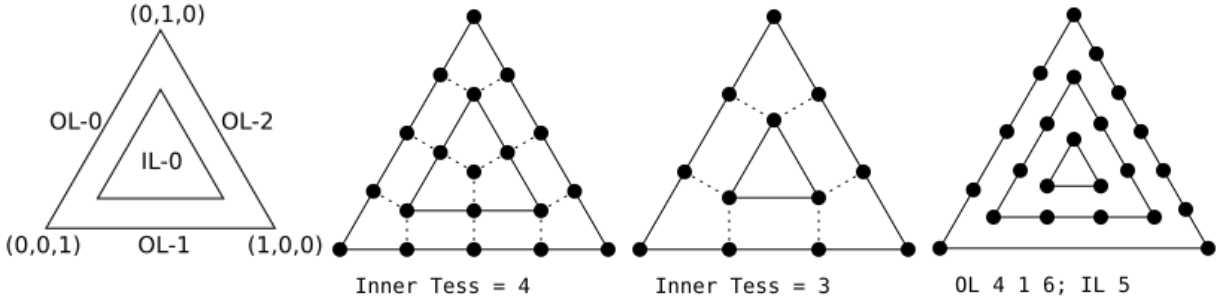


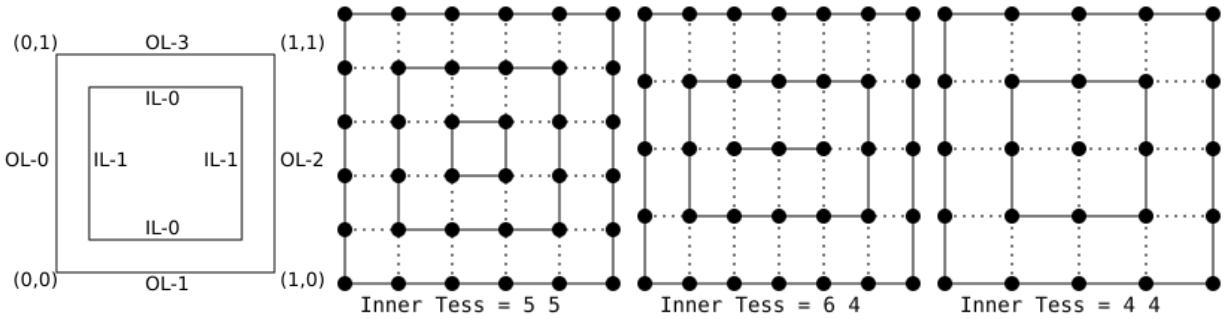
Quad Pattern

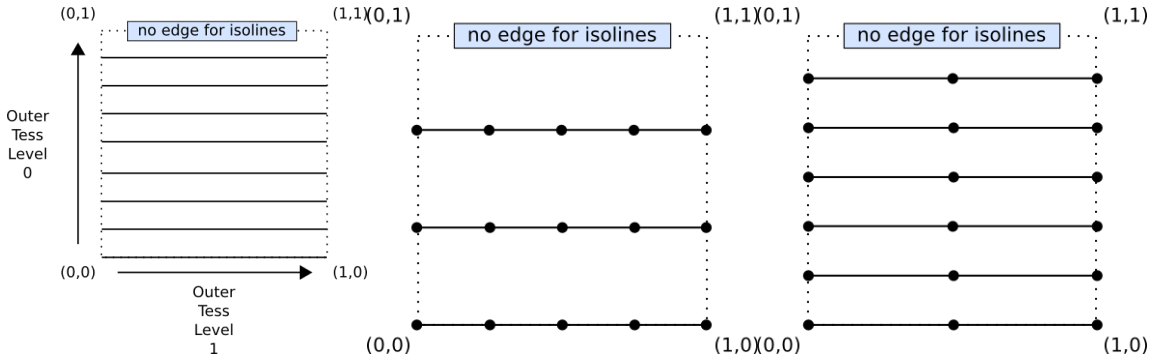


Isoline Pattern








[6]**Tessellation Evaluation Shader (TES)** – На входа му се подават абстрактните координати генерирани от TPG, заедно с координатите на възлите генерирани от TCS. Всички те се използват за генериране на крайните стойности за възлите. TES е необходим за създаване на tessellation – когато няма TES, процеса на tessellation изобщо не се случва. Колко пъти ще се извика TES зависи от различните имплементации, но се извиква най-малко толкова пъти, колкото има абстрактни координати генерирани от TPG.

**Вход**

in vec2 vertexTexCoord[]; - големината е като на входния patch

**Вградени** входни променливи

in vec3 gl_TessCoord; – локацията на абстрактния patch

in int gl_PatchVerticesIn; - броя на възли в обработвания patch

in int gl_PrimitiveID; - индекса на текущия patch

```
in gl_PerVertex{
        vec4 gl_Position;
        float gl_PointSize;
        float gl_ClipDistance[];
}gl_in[gl_MaxPatchVertices]; - входни променливи от TCS
```

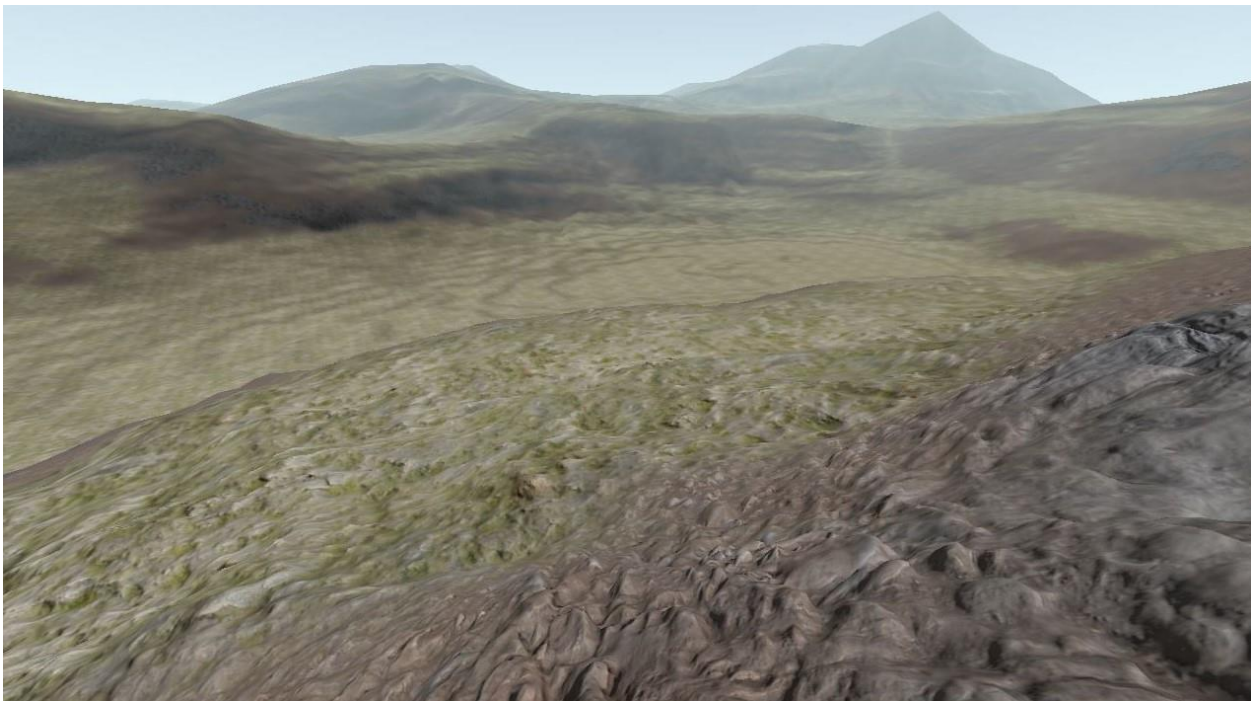
**Изход**

Могат да бъдат квалификатори за интерполация (flat, noperspective, smooth)

**Вградени** изходни променливи

```
out gl_PerVertex{
        vec4 gl_Position;
        float gl_PointSize;
        float gl_ClipDistance[];
};
```

## Terrain Demo



[9]В демото се използва библиотеката Lightweight Java Game Library (lwjgl). Тя поддържа всички версии на OpenGL, включително и последната спецификация - OpenGL 4.5.

Демото представлява генериране на терен и небе, като Tessellation Shader-и се използват само за генериране на терена. Със стрелките от клавиатурата се променя вектора на наблюдение на камерата, а с WASD контролите – позицията на камерата. Tessellation Shader-ите се добавят в "src/core/shaders/Shader.Java" по следния начин:

```
public void addTessellationControlShader(String text)
{
	addProgram(text, GL_TESS_CONTROL_SHADER);
}
public void addTessellationEvaluationShader(String text)
{
	addProgram(text, GL_TESS_EVALUATION_SHADER);
}
```

Стойността на параметъра text е пътя до мястото, където са дефинирани shader-ите. Тези две функции се викат от "src/modules/terrain/TerrainShader.java" в конструктора на класа.

```
addTessellationControlShader(ResourceLoader.loadShader("shaders/terrain/terrain_TC.glsl"));
addTessellationEvaluationShader(ResourceLoader.loadShader("shaders/terrain/terrain_TE.glsl"));
```

За добавяне на shader-а към програмата се използват функциите: int glCreateShader(int type), glShaderSource(int shader, CharSequence string), glCompileShader(int shader), glAttachShader(int program, int shader).

```
private void addProgram(String text, int type)
{
	int shader = glCreateShader(type);

	if (shader == 0)
	{
		System.err.println(this.getClass().getName() + " Shader creation failed");
		System.exit(1);
	}

	glShaderSource(shader, text);
	glCompileShader(shader);

	if(glGetShaderi(shader, GL_COMPILE_STATUS) == 0)
	{
		System.err.println(this.getClass().getName() + " " + glGetShaderInfoLog(shader, 1024));
		System.exit(1);
	}

	glAttachShader(program, shader);
}
```

В този конструктор също се добавят и [8]uniforms, които са глобални shader променливи. Те се отнасят като параметри, които потребителя може да подаде като параметри на shader програмата.

Те не се променят от едно извикване на shader-а до друго в рамките на една команда за изчертаване. Това ги прави различни от стандартните входове и изходи, които се променят на всяко извикване.

```
addUniform("tessellationFactor");
addUniform("tessellationSlope");
addUniform("tessellationShift");
```

Използват се за изчисляване нивото на tessellation – level of detail (LOD).

# Terrain Demo - TCS

```
#version 430

layout(vertices = 16) out;

in vec2 mapCoord_TC[];

out vec2 mapCoord_TE[];

const int AB = 2;
const int BC = 3;
const int CD = 0;
const int DA = 1;

uniform int tessellationFactor;
uniform float tessellationSlope;
uniform float tessellationShift;
uniform vec3 cameraPosition;

float lodFactor(float dist) {

	float tessellationLevel = max(0.0, tessellationFactor/pow(dist, tessellationSlope) +
tessellationShift);

	return tessellationLevel;
}

void main(){

	if (gl_InvocationID == 0){

		vec3 abMid = vec3(gl_in[0].gl_Position + gl_in[3].gl_Position)/2.0;
		vec3 bcMid = vec3(gl_in[3].gl_Position + gl_in[15].gl_Position)/2.0;
		vec3 cdMid = vec3(gl_in[15].gl_Position + gl_in[12].gl_Position)/2.0;
		vec3 daMid = vec3(gl_in[12].gl_Position + gl_in[0].gl_Position)/2.0;

		float distanceAB = distance(abMid, cameraPosition);
		float distanceBC = distance(bcMid, cameraPosition);
		float distanceCD = distance(cdMid, cameraPosition);
		float distanceDA = distance(daMid, cameraPosition);

		gl_TessLevelOuter[AB] = mix(1, gl_MaxTessGenLevel, lodFactor(distanceAB));
		gl_TessLevelOuter[BC] = mix(1, gl_MaxTessGenLevel, lodFactor(distanceBC));
		gl_TessLevelOuter[CD] = mix(1, gl_MaxTessGenLevel, lodFactor(distanceCD));
		gl_TessLevelOuter[DA] = mix(1, gl_MaxTessGenLevel, lodFactor(distanceDA));

		gl_TessLevelInner[0] = (gl_TessLevelOuter[BC] + gl_TessLevelOuter[DA])/4;
		gl_TessLevelInner[1] = (gl_TessLevelOuter[AB] + gl_TessLevelOuter[CD])/4;
	}

	mapCoord_TE[gl_InvocationID] = mapCoord_TC[gl_InvocationID];
	gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
}
```

# Terrain Demo - TES

```
#version 430

layout(quads, fractional_odd_spacing, cw) in;

in vec2 mapCoord_TE[];
out vec2 mapCoord_GS;
uniform sampler2D heightmap;

void main(){

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

        // world position
        vec4 position =
        ((1 - u) * (1 - v) * gl_in[12].gl_Position +
        u * (1 - v) * gl_in[0].gl_Position +
        u * v * gl_in[3].gl_Position +
        (1 - u) * v * gl_in[15].gl_Position);

        vec2 mapCoord =
        ((1 - u) * (1 - v) * mapCoord_TE[12] +
        u * (1 - v) * mapCoord_TE[0] +
        u * v * mapCoord_TE[3] +
        (1 - u) * v * mapCoord_TE[15]);

        float height = texture(heightmap, mapCoord).r;
        height *= 600;

        position.y = height;

        mapCoord_GS = mapCoord;

        gl_Position = position;
}
```

# Алтернативи на TCS и TES – също използващи се при генериране на терен

## TCS

```
#version 450 core

//how many times this shader will run per patch (n of out control points)
layout (vertices = 4) out;

in VertexShaderOut {
	vec2 texCoord;
	vec2 texCoord2;
} tcs_in[];

out TCShaderOut {
	vec2 texCoord;
	vec2 texCoord2;
} tcs_out[];

layout(location = 11) uniform mat4 view;
layout(location = 13) uniform mat4 projection;
layout(location = 14) uniform vec4 frustumPlanes[6];

layout (binding = 0) uniform sampler2D displacementTexture;

//desired pixels per tri edge
layout(location = 22) uniform float tessellatedTriWidth;
layout(location = 23) uniform ivec2 viewportSize;
layout(location = 24) uniform float displacementScale;

////////////////////////////////
//FRUSTUM CULLING
////////////////////////////////
struct AABB {
	vec3 min, max;
};

bool planeAABBIntersection(AABB aabb, vec4 plane) {
	vec3 halfDiagonal =  (aabb.max - aabb.min) * 0.5;
	vec3 center = aabb.min + halfDiagonal;

	float e = halfDiagonal.x + abs(plane.x) +
				halfDiagonal.y + abs(plane.y) +
				halfDiagonal.z + abs(plane.z);

	float s = dot(center, plane.xyz) + plane.w;

	if(s - e > 0.0) return false; //outside
	else if(s + e < 0.0) return true; //inside
	else return true; //intersecting
}

bool frustumAABBIntersection(AABB aabb) {
	bool intersection = false;
	for(int i = 0; i < 6; ++i) {
		bool result = planeAABBIntersection(aabb, frustumPlanes[i]);
		if(!result) return false;
		else intersection = true;
	}
	return intersection;
}

void createPatchAABB(vec3 p1, vec3 p2, vec3 p3, vec3 p4, out AABB aabb) {
	//Using knowledge of the input vertices
	aabb.min.x = p4.x;
	aabb.min.y = min(0.0, displacementScale);
	aabb.min.z = p4.z;

	aabb.max.x = p2.x;
```

```
		aabb.max.y = max(0.0, displacementScale);
		aabb.max.z = p2.z;
}

////////////////////////////////
//LOD COMPUTATION
////////////////////////////////
// Project a sphere into clip space and return the number of triangles that are required
// to fit across the screenspace diameter.
float sphereToScreenSpaceTessellation(vec3 cp0, vec3 cp1, float diameter) {
		vec3 mid = (cp0 + cp1) * 0.5;

		vec4 p0 = view * vec4(mid, 1.0);
		vec4 p1 = p0;
		p1.x += diameter;

		vec4 clip0 = projection * p0;
		vec4 clip1 = projection * p1;

		clip0 /= clip0.w;
		clip1 /= clip1.w;

		clip0.xy = ((clip0.xy * 0.5) + 0.5) * viewportSize;
		clip1.xy = ((clip1.xy * 0.5) + 0.5) * viewportSize;

		return distance(clip0, clip1) / tessellatedTriWidth; //spec says it will be clamped by
GL
		//return clamp(distance(clip0, clip1) / tessellatedTriWidth, 1.0, 64.0);
}

void main() {
		if(gl_InvocationID == 0) {

			AABB aabb;
			createPatchAABB(gl_in[0].gl_Position.xyz,
								gl_in[1].gl_Position.xyz,
								gl_in[2].gl_Position.xyz,
								gl_in[3].gl_Position.xyz,
								aabb);

			if(frustumAABBIntersection(aabb)) {
				//Copy control points positions to mess with their y without passing
to the pipeline
				vec4 p0 = gl_in[0].gl_Position;
				vec4 p1 = gl_in[1].gl_Position;
				vec4 p2 = gl_in[2].gl_Position;
				vec4 p3 = gl_in[3].gl_Position;

				//Displace control points on y, to account for height on LOD
computations
				p0.y += texture(displacementTexture, tcs_in[0].texCoord).r *
displacementScale;
				p1.y += texture(displacementTexture, tcs_in[1].texCoord).r *
displacementScale;
				p2.y += texture(displacementTexture, tcs_in[2].texCoord).r *
displacementScale;
				p3.y += texture(displacementTexture, tcs_in[3].texCoord).r *
displacementScale;

				/*const float maxX = max(abs(p1.x - p0.x), abs(p1.x - p2.x));
				const float maxY = max(abs(p1.y - p0.y), abs(p1.y - p2.y));
				const float maxZ = max(abs(p1.z - p0.z), abs(p1.z - p2.z));
				const float sideLen = max(max(maxX, maxY), maxZ);*/
				//const float sideLen = max(distance(p1, p0), distance(p1, p2));
				const float sideLen = p1.x - p0.x; //abusing knowing the tiles are
square

				gl_TessLevelOuter[0] = sphereToScreenSpaceTessellation(p3.xyz,
p0.xyz, sideLen);
				gl_TessLevelOuter[1] = sphereToScreenSpaceTessellation(p0.xyz,
p1.xyz, sideLen);
```

```
                                gl_TessLevelOuter[2] = sphereToScreenSpaceTessellation(p1.xyz,
p2.xyz, sideLen);
                                gl_TessLevelOuter[3] = sphereToScreenSpaceTessellation(p2.xyz,
p3.xyz, sideLen);

                                gl_TessLevelInner[0] = (gl_TessLevelOuter[1] + gl_TessLevelOuter[3])
* 0.5;
                                gl_TessLevelInner[1] = (gl_TessLevelOuter[0], gl_TessLevelOuter[2])
* 0.5;
                        }
                        else {
                                gl_TessLevelOuter[0] = -1.0;
                                gl_TessLevelOuter[1] = -1.0;
                                gl_TessLevelOuter[2] = -1.0;
                                gl_TessLevelOuter[3] = -1.0;

                                gl_TessLevelInner[0] = -1.0;
                                gl_TessLevelInner[1] = -1.0;
                        }
                }

                tcs_out[gl_InvocationID].texCoord = tcs_in[gl_InvocationID].texCoord;
                tcs_out[gl_InvocationID].texCoord2 = tcs_in[gl_InvocationID].texCoord2;

                gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
}
```

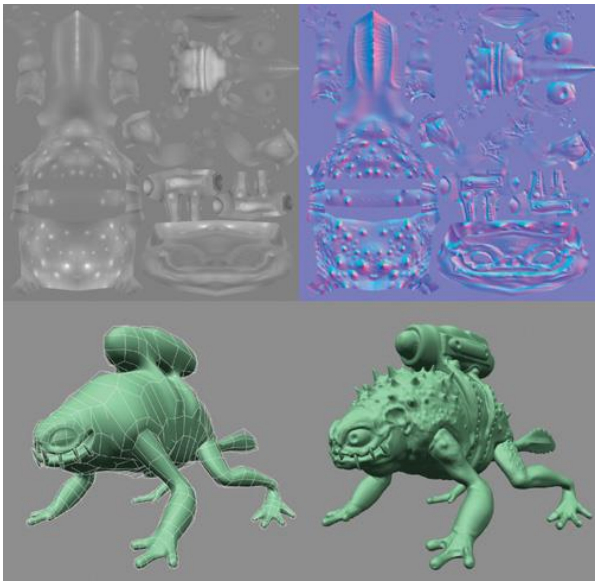
## TES

```
#version 450 core

//how the fixed tessellator will work, and how to interpret its output
layout (quads, fractional_even_spacing, ccw) in;

layout(location = 24) uniform float displacementScale;

layout (binding = 0) uniform sampler2D displacementTexture;

layout(location = 11) uniform mat4 view;

in TCShaderOut {
        vec2 texCoord;
        vec2 texCoord2;
} tes_in[];

out TEShaderOut {
        vec2 texCoord;
        vec2 texCoord2;
} tes_out;

void main() {
        vec2 t1 = mix(tes_in[0].texCoord, tes_in[1].texCoord, gl_TessCoord.x);
        vec2 t2 = mix(tes_in[3].texCoord, tes_in[2].texCoord, gl_TessCoord.x);
        tes_out.texCoord = mix(t1, t2, gl_TessCoord.y);

        vec2 t21 = mix(tes_in[0].texCoord2, tes_in[1].texCoord2, gl_TessCoord.x);
        vec2 t22 = mix(tes_in[3].texCoord2, tes_in[2].texCoord2, gl_TessCoord.x);
        tes_out.texCoord2 = mix(t21, t22, gl_TessCoord.y);

        vec4 p1 = mix(gl_in[0].gl_Position, gl_in[1].gl_Position, gl_TessCoord.x);
        vec4 p2 = mix(gl_in[3].gl_Position, gl_in[2].gl_Position, gl_TessCoord.x);
        gl_Position = mix(p1, p2, gl_TessCoord.y);
        gl_Position.y += texture(displacementTexture, tes_out.texCoord).r * displacementScale;
}
```

## Изводи

С помощ на Tessellation Shader-ите се увеличава качеството на крайното изображение. По-близките до камерата обекти или части от изображението са представени в по-големи нива на детайлност. Обекти (части), които са по-далеч от камерата не се изчисляват до толкова голяма степен. С това се запазват много от изчислителните ресурси.

Те намират приложения във филмовата индустрия, игрите, архитектурата, а често се срещат и при генериране на терени, панорами, силуети и т.н.



# Източници


1. https://en.wikipedia.org/wiki/Shader
2. https://www.haroldserrano.com/blog/what-is-tessellation-in-opengl
3. http://web.engr.oregonstate.edu/~mjb/cs519/Handouts/tessellation.1pp.pdf
4. https://www.khronos.org/opengl/wiki/Tessellation
5. https://www.khronos.org/opengl/wiki/Tessellation_Control_Shader
6. https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader
7. https://en.wikipedia.org/wiki/OpenGL_Shading_Language
8. https://www.khronos.org/opengl/wiki/Uniform_(GLSL)
9. https://www.youtube.com/watch?v=JaCQGIqxgZk&t=9s