**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"**

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота з дисципліни

“Візуалізація графічної та геометричної інформації”

на тему “Операції з текстурними координатами”

Виконав студент групи ТР-31мп

Чупров Владислав Вікторович

Київ - 2023

1. **Завдання**

* Нанести текстуру на поверхню з практичного завдання №2.
* Реалізувати масштабування/обертання текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки – непарні варіанти реалізують масштабування, парні варіанти реалізують обертання.
* Розробити функціонал для переміщення точки вздовж простору поверхні (u,v) за допомогою клавіатури. наприклад клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

1. **Теорія**

WebGL

WebGL — це інструмент JavaScript для створення динамічної 2D і 3D графіки прямо у веб-переглядачі. Використовуючи WebGL API, можна втілювати візуалізації з текстурами, масштабуваннями, обертаннями та інтерактивним переміщенням об’єктів, завдяки можливостям обробки шейдерів, текстур та матричних перетворень. • Шейдери служать для встановлення способу обробки вершин і фрагментів, уможливлюючи включення текстурних координат і різноманітних перетворень. • Для роботи з текстурами створюються спеціалізовані текстурні об’єкти, які потім прив’язуються до визначених текстурних одиниць, що дозволяє використовувати їх у шейдерах. • Матричні перетворення використовуються для ефективного керування позиціонуванням, розміром та орієнтацією 3D моделей.

Шейдери WebGL

У WebGL шейдери є ключовим елементом в процесі візуалізації, надаючи гнучкість і програмованість у формуванні зовнішнього вигляду 3D-графіки. Існують два основних типи шейдерів у WebGL: вершинні та фрагментні.

Вершинні шейдери відіграють роль у обробці кожної вершини 3D-моделі перед її проектуванням на екран. Вони отримують атрибути з даних вершин, такі як їхнє положення, колір, координати текстури, та обробляють ці дані для створення кінцевого результату. Вершинні шейдери здійснюють різні трансформації на позиціях вершин, включаючи переміщення, обертання та масштабування, що є критичними для розміщення об'єктів у тривимірному просторі. Результати роботи вершинного шейдера передаються до фрагментного шейдера, включаючи перетворене положення вершини та будь-які інші інтерпольовані значення.

Фрагментні шейдери, іноді називані піксельними шейдерами, відповідають за обробку кожного пікселя на екрані. Вони отримують інтерпольовані значення від вершинного шейдера, включаючи координати кольору та текстури, а також додаткову інформацію, наприклад, про освітлення. Основна функція фрагментного шейдера - це визначення кінцевого кольору пікселя, що може включати вибірку текстури, розрахунки освітлення та інші ефекти. Кінцевий колір використовується для візуалізації пікселя на екрані.

Як вершинні, так і фрагментні шейдери пишуться на мові GLSL (OpenGL Shading Language) і вимагають компіляції перед їх використанням. Після компіляції ці шейдери об'єднуються у програму шейдера, яка повинна бути зв'язана з WebGL перед запуском рендеринга.

І вершинний, і фрагментний шейдери написані мовою GLSL (OpenGL Shading Language) і їх потрібно скомпілювати, перш ніж їх можна буде використовувати. Після компіляції вони об’єднуються в програму шейдера. Програма шейдера повинна бути зв’язана перед відтворенням, дозволяючи WebGL використовувати вказані вершинні та фрагментні шейдери для конвеєра відтворення.

Uniforms – це значення, які залишаються постійними для всіх вершин або фрагментів під час візуалізації примітиву. Вони дозволяють передавати шейдерам зовнішні дані, такі як матриці трансформації або інформацію про глобальне освітлення.

Attributes — це дані для кожної вершини, які відрізняються між вершинами. Вони використовуються для передачі такої інформації, як положення вершин, нормалі та координати текстури.

Координати текстури зазвичай передаються як атрибути до вершинного шейдера, а потім інтерполюються через примітив для використання у фрагментному шейдері. У фрагментному шейдері координати текстури використовуються для вибірки кольорів із текстур. Вибрані кольори потім використовуються для визначення остаточного кольору кожного пікселя.

Накладання текстур. Текстурні координати

Відображення текстури є ключовою технікою у комп'ютерній графіці, що дозволяє досягнути реалістичного вигляду поверхонь шляхом проєктування текстурних зображень на 3D-моделі. У рамках WebGL, ця техніка включає присвоєння кожній вершині 3D-об'єкта відповідних координат текстури (u, v). Ці координати використовуються для визначення, які частини текстурного зображення мають бути нанесені на конкретні частини моделі. Такий підхід значно підвищує реалізм і деталізацію зовнішнього вигляду об'єктів, надаючи їм більш точне та натуралістичне текстурне покриття.

Координати текстури (u, v) є важливими параметрами, які визначають спосіб нанесення текстури на поверхню. Вони варіюються від 0 до 1 і використовуються для звернення до конкретних точок на зображенні текстури. У контексті WebGL координати текстури пов’язані з кожною вершиною 3D-моделі та інтерполюються по всій поверхні під час візуалізації. Це гарантує, що текстура точно охоплює модель.

1. **Деталі розробки**

За варіантом мені було надано Virich Cyclic Surface. При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

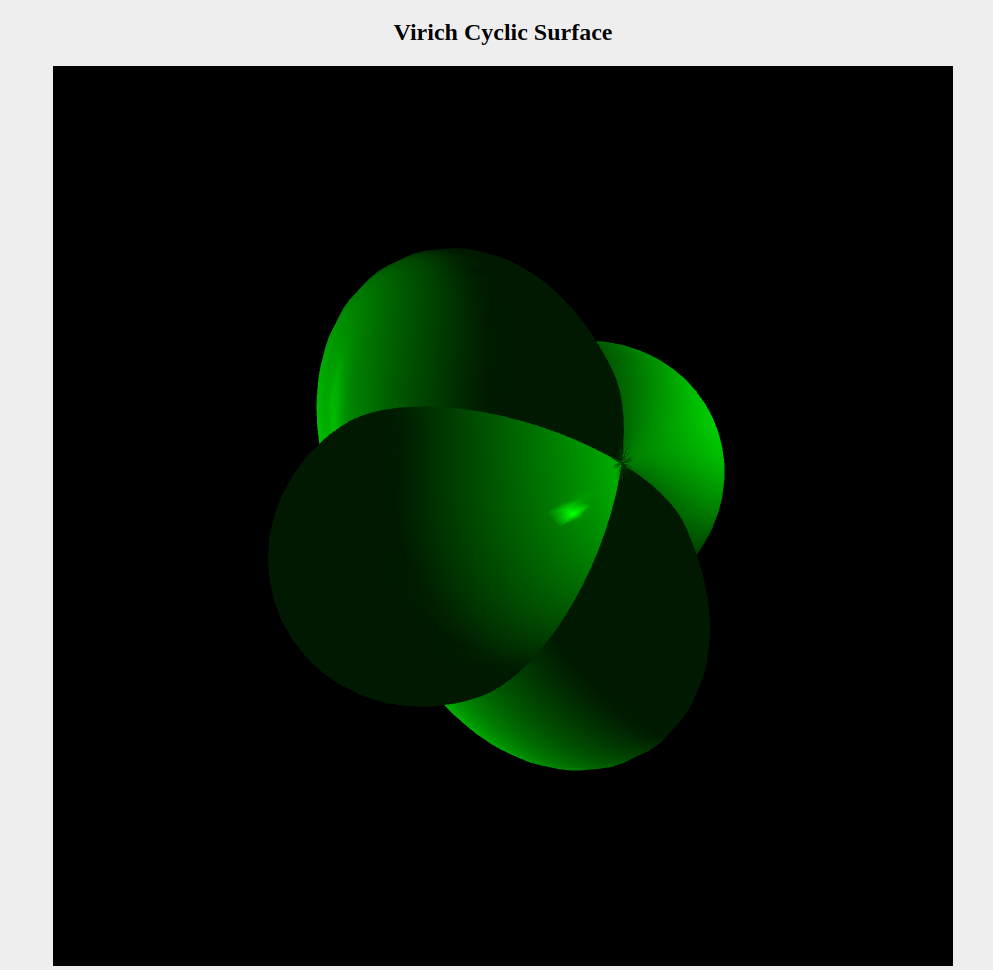


Рисунок 1 - Вигляд поверхні

Я обрав наступне зображення для подальшого виконання розрахунково-графічної роботи. Для підтримки на більшій кількості операційних систем та браузерів було обрано зображення розміром 512х512 пікселів. Формат зображення .bmp.

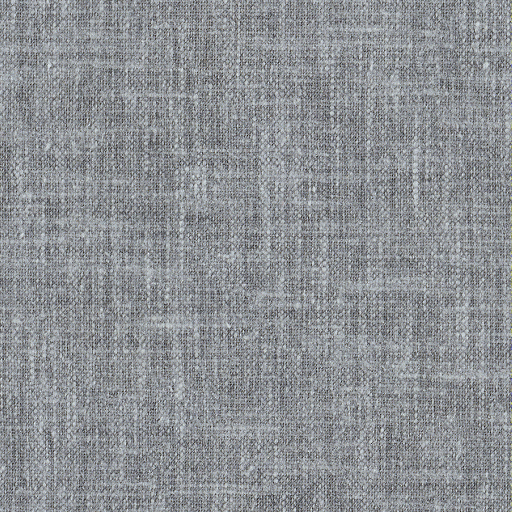


Рисунок 2 - Зображення текстури

Було накладено текстуру на поверхню. Для накладання текстури на поверхню було підготовлено буфер текстурних координат, кожна з яких відповідає елементу масива з буферу вершин. Згідно варіанту текстура має масштабуватися, тому було створено відповідний uniform, який визначатиме фактор масштабування. В програмі шейдера для визначення кольору пікселя фігури на яку накладено текстуру необхідно використовувати функцію texture2D(). Ця функція першим аргументом приймає обʼєкт класу sampler2D, який зберігає дані про зображення, а другим текстурну координату.

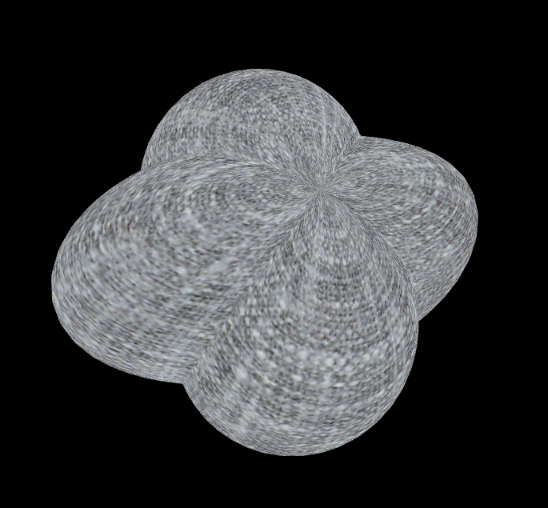


Рисунок 3 - Текстура накладена на поверхню

Було створено новий обʼєкт класу Model для відображення точки відносно якої буде здійснюватись трансформація текстури. Обʼєкт графічно відображається як сфера. Сфера перебуває на поверхні.

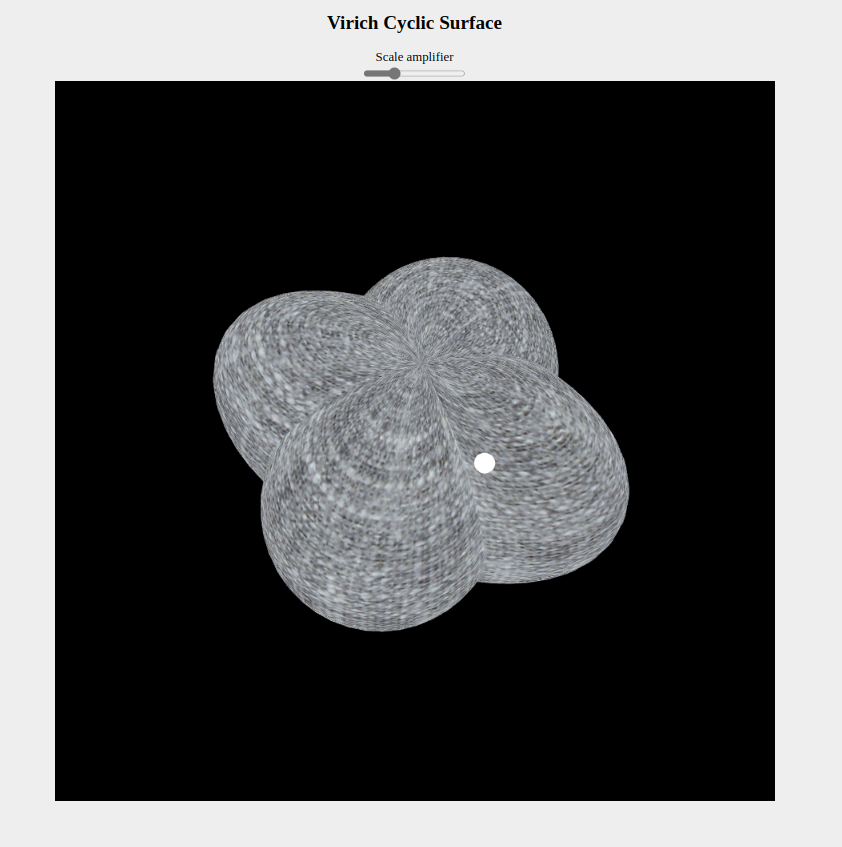


Рисунок 4 - Точка на текстурованій поверхні

Згідно варіанту було імплементовано масштабування текстури відносно точки на поверхні.

1. **Інструкції користувача**

Фігуру можна обертати відносно центру затиснувши ліву клавішу миші та потягнувши в сторону бажаного обертання.

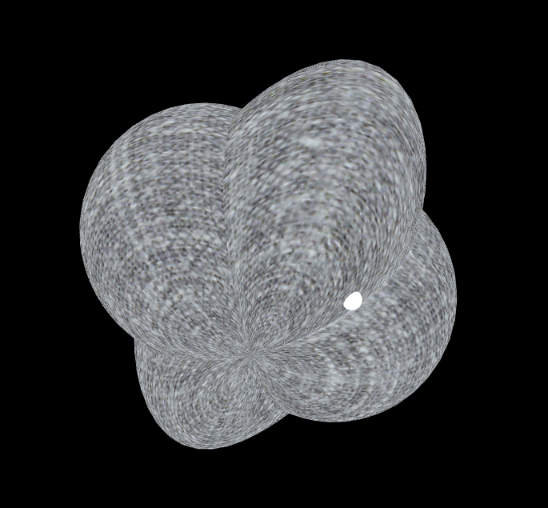
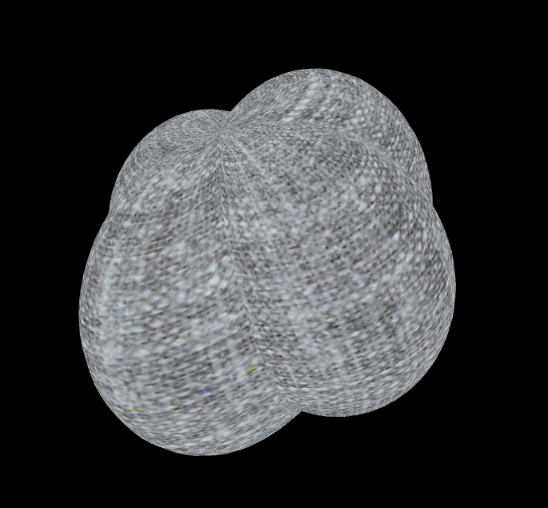


Рисунок 5 - Вигляд фігури до та після обертання

Переміщувати точку відносно якої здійснюється обертання можна за допомогою клавіш WASD. Кожне натискання переміщує точку по поверхні на визначений крок. Переміщення здійснюється до визначеної межі.

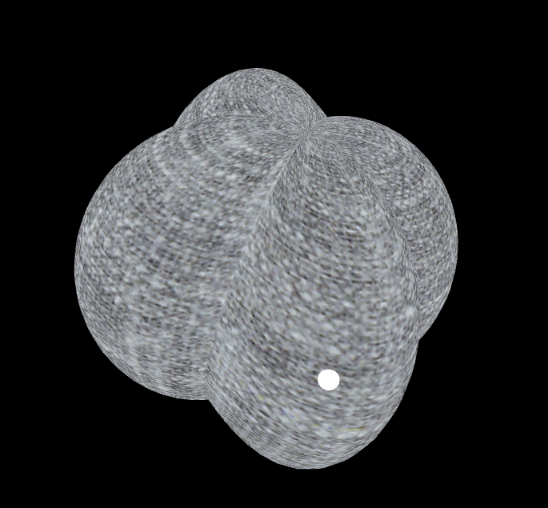
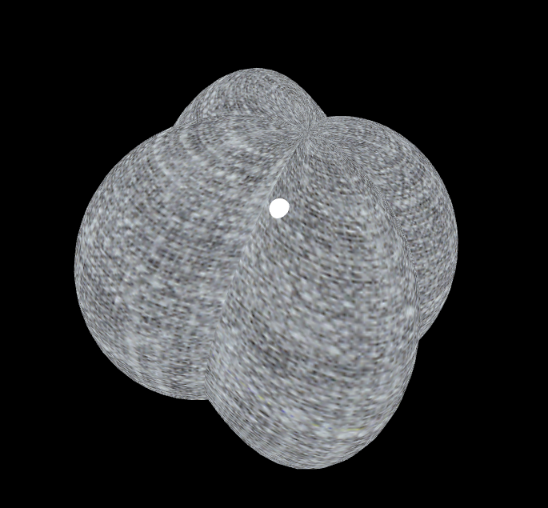


Рисунок 6 - Демонстрація переміщення точки відносно нерухомої фігури

За допомогою слайдера з підписом “Scale amplifier” можна змінювати фактор масштабування відносно умовної точки на поверхні.

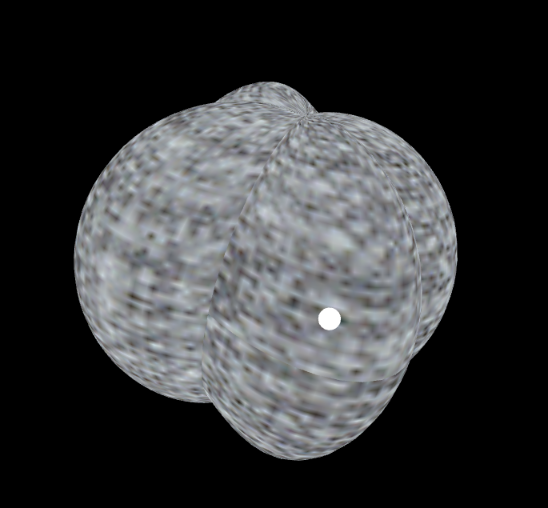
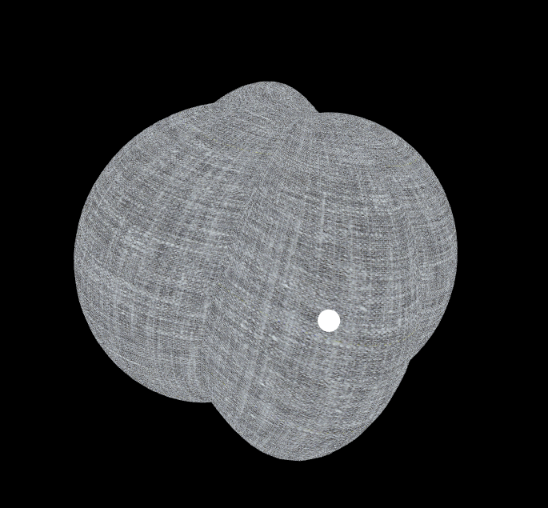


Рисунок 7 - Демонстрація зміни фактору масштабування текстури

При перезавантаженні сторінки обертання фігури, а також значення фактору масштабування буде встановлено по замовчуванню. Положення точки відносно поверхні також буде скинуто до значення по замовчуванню.

1. **Код програми**

Код керування точки

*window*.onkeydown = (e) => {  
 if (e.keyCode == 87) {  
 *spherePos*[0] = *Math*.min(*spherePos*[0] + 0.1, *Math*.PI \* 2);  
 }  
 else if (e.keyCode == 83) {  
 *spherePos*[0] = *Math*.max(*spherePos*[0] - 0.1, 0);  
 }  
 else if (e.keyCode == 68) {  
 *spherePos*[1] = *Math*.min(*spherePos*[1] + 0.1, 2 \* *Math*.PI);  
 }  
 else if (e.keyCode == 65) {  
 *spherePos*[1] = *Math*.max(*spherePos*[1] - 0.1, 0);  
 }  
}

Створення буферу для вершин текстури

function Model(name) {  
 this.name = name;  
 this.iVertexBuffer = *gl*.createBuffer();  
 this.iNormalBuffer = *gl*.createBuffer();  
 this.iTexCoordsBuffer = *gl*.createBuffer();  
 this.count = 0;  
  
 this.BufferData = function (vertices, normals, texCoords) {  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(vertices), *gl*.*STREAM\_DRAW*);  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iNormalBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(normals), *gl*.*STREAM\_DRAW*);  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iTexCoordsBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(texCoords), *gl*.*STREAM\_DRAW*);  
 this.count = vertices.length / 3;  
 };  
 this.Draw = function () {  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribVertex, 3, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribVertex);  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iNormalBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribNormal, 3, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribNormal);  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iTexCoordsBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribTexCoord, 2, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribTexCoord);  
  
 // gl.drawArrays(gl.LINE\_STRIP, 0, this.count);  
 *gl*.drawArrays(*gl*.*TRIANGLES*, 0, this.count);  
 };  
}

fragment shader

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH  
precision highp float;  
#else  
precision mediump float;  
#endif  
  
varying vec3 varCol;  
uniform vec4 color;  
uniform sampler2D tmu;  
varying vec2 texCoordV;  
  
void main() {  
 vec4 textureColor = texture2D(tmu, texCoordV);  
 gl\_FragColor = textureColor \* vec4(varCol, 1.0);  
  
 if (color.a > 100.0) {  
 gl\_FragColor = vec4(1.0);  
 }  
}

Vertex shader

attribute vec3 vertex, normal;  
attribute vec2 texCoord;  
uniform mat4 ModelViewProjectionMatrix, NormalM;  
varying vec3 varCol;  
varying vec2 texCoordV;  
uniform vec3 texTranslate;  
uniform float scaleAmpl;  
mat4 translation(vec3 t) {  
 return mat4(  
 vec4(1.0, 0.0, 0.0, 0.0),  
 vec4(0.0, 1.0, 0.0, 0.0),  
 vec4(0.0, 0.0, 1.0, 0.0),  
 vec4(t, 1.0)  
 );  
}  
mat4 scaling(float s) {  
 return mat4(  
 vec4(s, 0.0, 0.0, 0.0),  
 vec4(0.0, s, 0.0, 0.0),  
 vec4(0.0, 0.0, s, 0.0),  
 vec4(0.0, 0.0, 0.0, 1.0)  
 );  
}  
void main() {  
 vec4 texTransformed = translation(texTranslate) \* scaling(scaleAmpl) \* translation(-texTranslate) \* vec4(texCoord, 0.0, 1.0);  
 texCoordV = texTransformed.xy;  
 gl\_Position = ModelViewProjectionMatrix \* vec4(vertex, 1.0);  
 varCol = vec3(1.0); // Предположим, что цвет по умолчанию - белый  
}