Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. І. Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

з курсу: «Візуалізація графічної та геометричної інформації»

на тему: «Операції з технічного узгодження»

Варіант №22

**Виконав:**

студент групи ТР-31мп

Плаксицький Всеволод Андрійович

Київ 2023

**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.
* Створити можливість 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.

WebGL (Web Graphics Library) - це технологія, яка дозволяє вбудовувати 3D та 2D графіку безпосередньо в веб-браузери. Вона базується на OpenGL ES (Embedded Systems), що є відкритим стандартом для відтворення графіки на вбудованих системах, таких як мобільні телефони та інші пристрої. WebGL дає можливість використовувати мову програмування JavaScript для написання програм, які керують графічними об'єктами на веб-сторінці. Це дозволяє створювати 3D-анімації та інші візуальні ефекти без необхідності встановлення додаткових плагінів чи розширень.

Для втілення відображення текстур, масштабування, обертання та інтерактивного переміщення точок у WebGL API використовуються функції для роботи з шейдерами, текстурами та матричними перетвореннями.

Шейдери використовуються для визначення обробки вершин та фрагментів, що дозволяє включити текстурні координати та здійснювати перетворення.

Текстурні об'єкти створюються та прив'язуються до конкретних текстурних одиниць для використання у шейдерах.

Матричні перетворення використовуються для управління положенням, масштабуванням і обертанням 3D-моделі.

У WebGL, шейдери використовуються для програмного керування обробкою вершин і фрагментів графічних об'єктів. Шейдери написані мовою GLSL (OpenGL Shading Language) і виконуються на графічному процесорі (GPU). WebGL використовує два основних типи шейдерів: вершинні шейдери (vertex shaders) і фрагментні шейдери (fragment shaders).

Вершинні шейдери відповідають за обробку кожної вершини 3D-моделі перед її відображенням на екрані. Ці шейдери отримують атрибути з даних вершин, таких як положення, колір і координати текстури, і обробляють їх для створення вихідних даних. Одним із популярних способів використання вершинних шейдерів є застосування трансформацій до позицій вершин. Це включає переміщення, обертання та масштабування, що є ключовими для позиціонування об'єктів у 3D-просторі. Вихідні дані вершинного шейдера включають у себе трансформовану позицію вершини та будь-які інтерпольовані значення, які передаються фрагментному шейдеру.

Фрагментні шейдери працюють з кожним пікселем, який буде відображений на екрані. Вони отримують інтерпольовані значення від вершинного шейдера, такі як координати кольору та текстури, а також інші дані, наприклад, інформацію про освітлення. Основне завдання фрагментного шейдера - визначити остаточний колір для кожного пікселя. Це може включати вибірку текстури, обчислення освітлення та інші ефекти. Кінцевий вихідний колір використовується для малювання пікселя на екрані.

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

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

Відображення текстури - це техніка в комп'ютерній графіці, яка дозволяє створювати реалістичні поверхні, накладаючи зображення або текстури на 3D-моделі. У контексті WebGL відображення текстур включає асоціацію кожної вершини 3D-об'єкта з координатами текстури (u, v), які потім використовуються для вибору кольорів із зображення текстури. Це покращує візуальний вигляд об'єкта, надаючи детальну інформацію про його поверхню. Координати текстури (u, v) є важливими параметрами, які визначають, як текстура накладається на поверхню. Вони варіюються від 0 до 1 і використовуються для звернення до конкретних точок на текстурі. 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.

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 вибору кольорів із текстури. Вибрані кольори потім використовуються для визначення остаточного кольору кожного пікселя.

При виконанні практичного завдання №2 було розроблено програму, що виводить аналітичну поверхню – Параболічний Гуммінг-Топ (Parabolic Humming-Top) у вигляді суцільних трикутників для відображення її форми та структури.

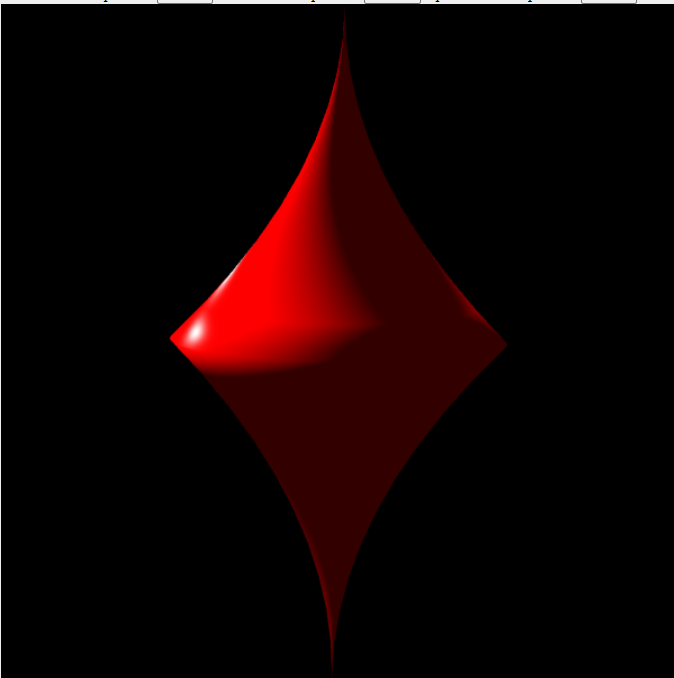


Рисунок 1 – Вигляд поверхні Parabolic Humming-Top

Зображення текстури у форматі .jpg, що була обрана для накладання на поверхню, використовується для подальших графічних операцій та візуалізації.



Рисунок 2 – Зображення текстури

Для накладання текстури на поверхню було підготовлено буфер текстурних координат, де кожна координата відповідає елементу масиву з буферу вершин. Згідно варіанту, текстура має обертатися, тому було створено відповідний uniform, який визначатиме кут обертання. У програмі шейдера для визначення кольору пікселя фігури, на яку накладено текстуру, використовувалась функція texture2D(). Ця функція приймає об'єкт класу sampler2D як перший аргумент, що містить дані про зображення, і другий аргумент - текстурну координату. Такий підхід дозволяє управляти текстурою, забезпечуючи можливість обертання відповідно до заданого кута обертання за допомогою відповідного uniform.

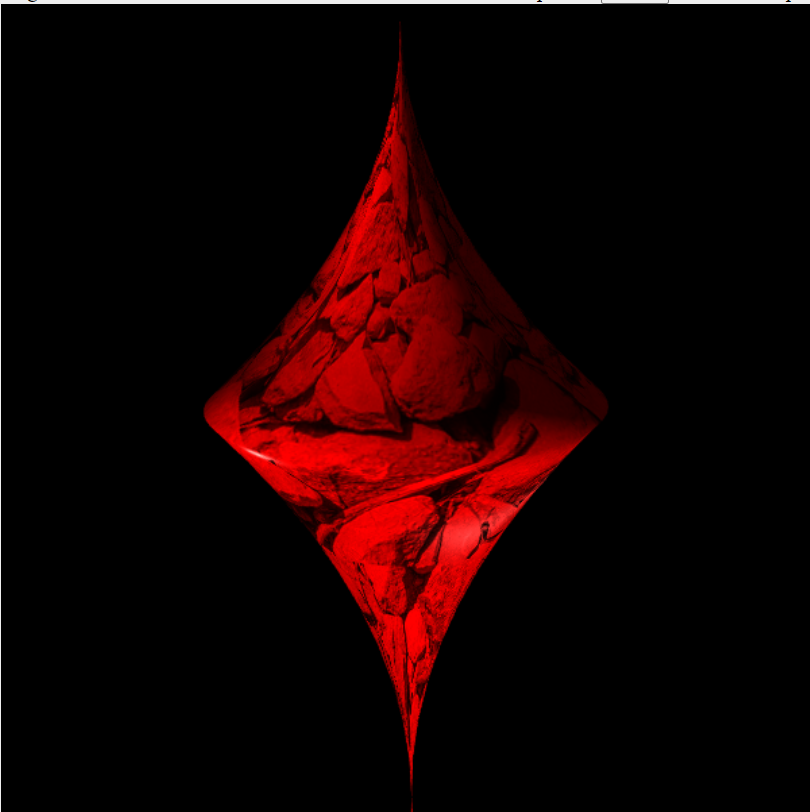


Рисунок 3 – Текстура накладена на поверхню

Було створено новий об'єкт класу Model для відображення точки, відносно якої буде здійснюватись трансформація текстури. Графічне відображення об'єкта подано у вигляді сфери, яка розташована на поверхні.

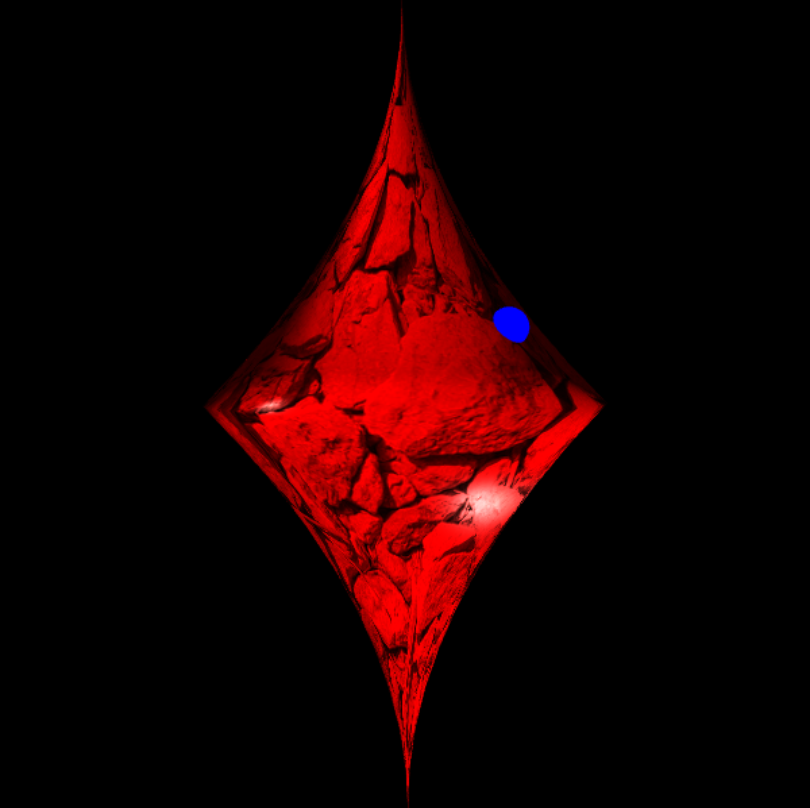


Рисунок 4 – Точка на текстурованій поверхні

**Інструкції користувача**

Фігуру можна обертати, тримаючи ліву кнопку миші та пересуваючи її в бажаному напрямку. Також є можливість змінювати модель освітлення, підсумовуючи вплив навколишнього, дифузного та дзеркального компонентів.

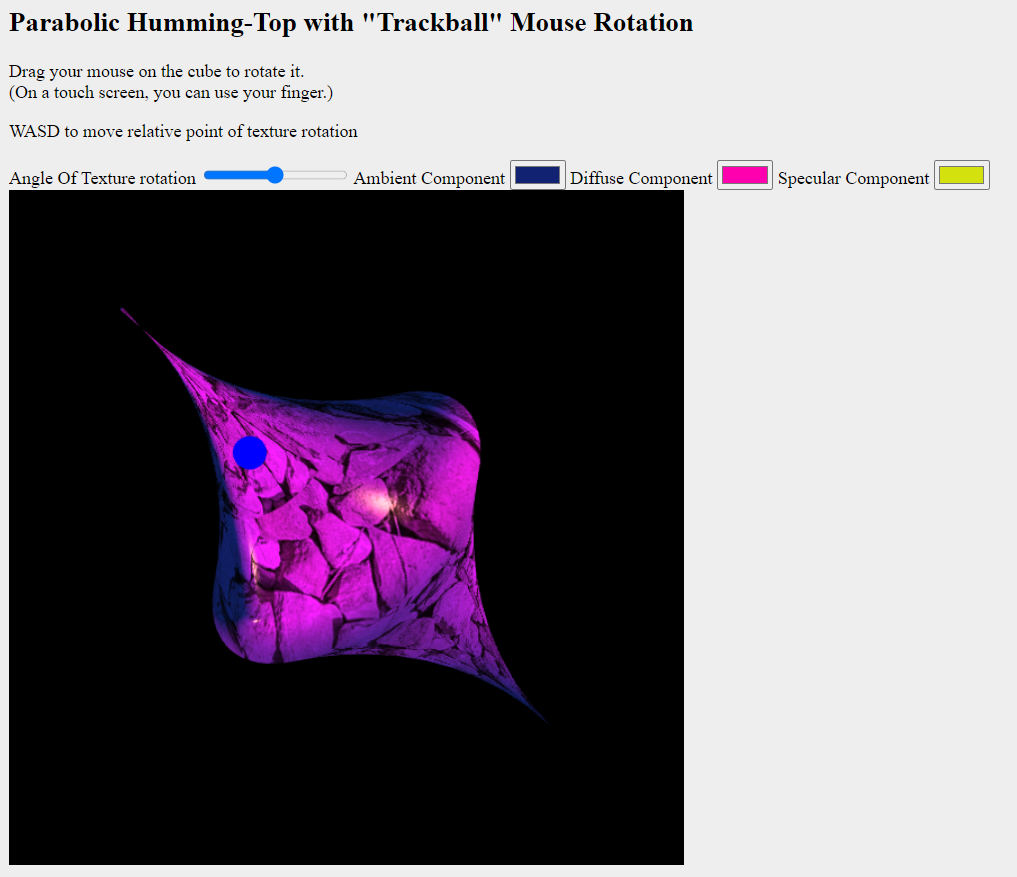


Рисунок 5 – Демонстрація можливості обертання та змінення кольору фігури

Переміщувати точку центра обертання можна за допомогою клавіш W, A, S, D. Кожне натискання на кнопку здійснює переміщення точки по поверхні на визначений крок, що продовжується до досягнення визначеної межі.

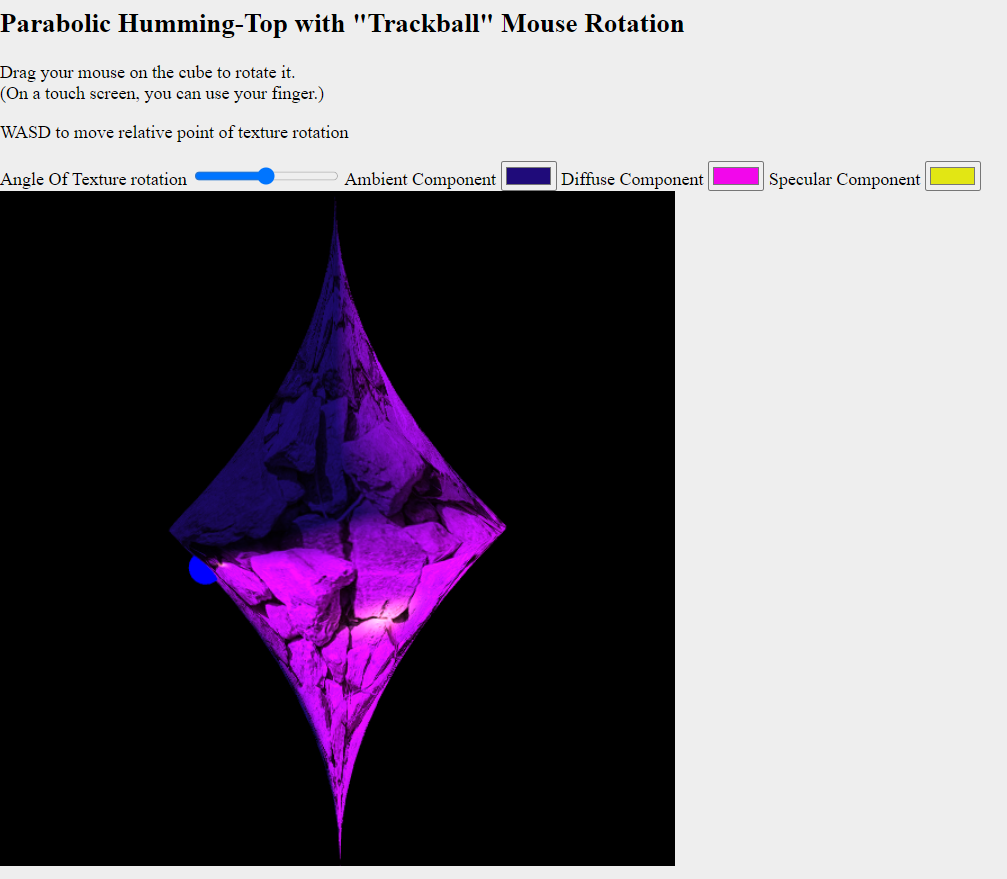


Рисунок 6 – Демонстрація можливості переміщення точки центра обертання

Використовуючи слайдер з підписом "Angle Of Texture rotation", можна змінювати кут обертання відносно визначеної точки на поверхні. Якщо значення кута не дорівнює нулю, при переміщенні точки також зміщується текстура. Це відбувається через те, що обертання відбувається відносно іншої точки на поверхні, яка взаємодіє з іншою текстурною координатою.

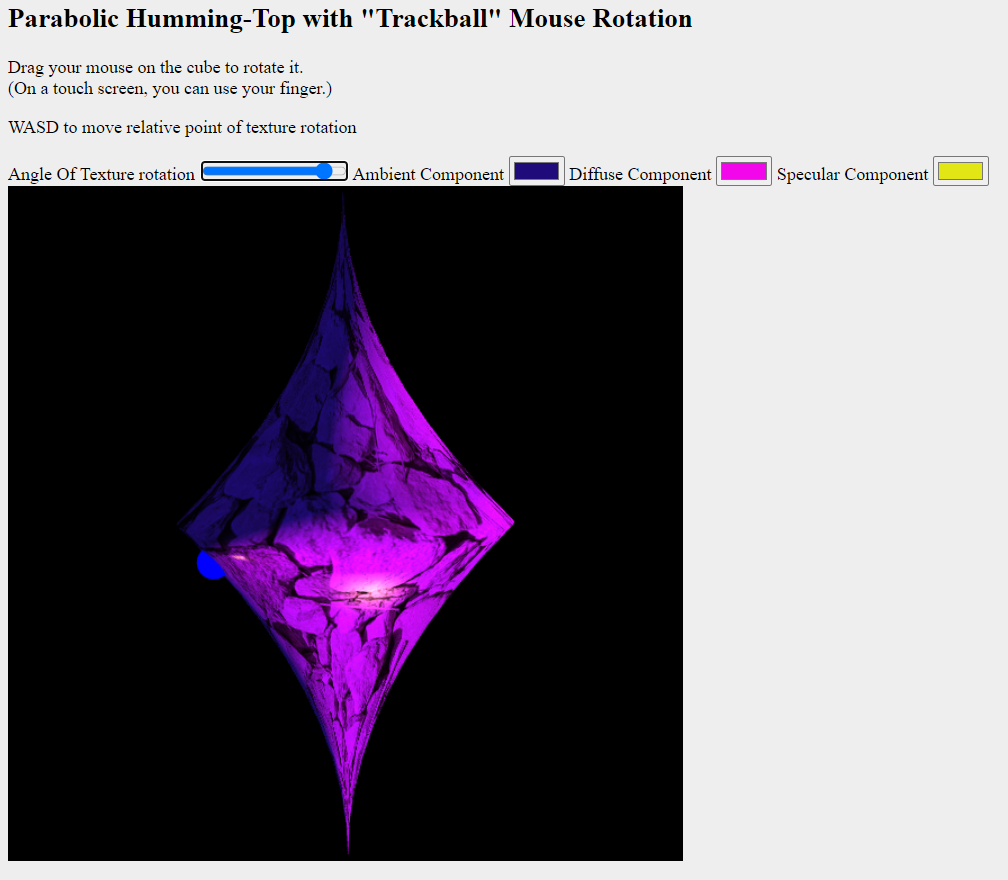
****

Рисунок 7 – Демонстрація можливості зміни кута обертання текстури

Після перезавантаження сторінки положення обертання фігури, колір фігури, значення кута обертання та положення точки відносно поверхні будуть встановлені в значення за замовчуванням.

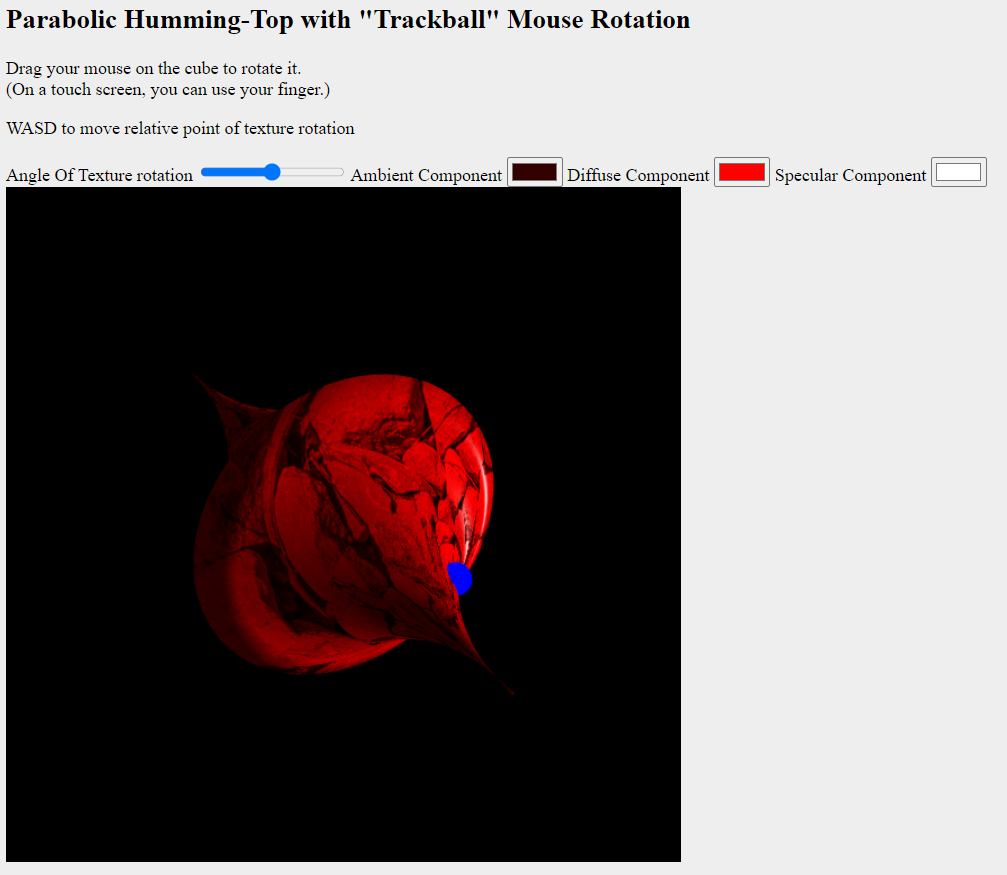
****

Рисунок 8 – Демонстрація фігури при значень за замовчуванням

**Код програми**

function draw() {

gl.clearColor(0,0,0,1);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

/\* Set the values of the projection transformation \*/

let projection = m4.orthographic(-1, 1, -1, 1, -1, 1);

/\* Get the view matrix from the SimpleRotator object.\*/

let modelView = spaceball.getViewMatrix();

let rotateToPointZero = m4.axisRotation([0.707,0.707,0], 0.7);

let translateToPointZero = m4.translation(0,0,0);

let matAccum0 = m4.multiply(rotateToPointZero, modelView );

let matAccum1 = m4.multiply(translateToPointZero, matAccum0 );

/\* Multiply the projection matrix times the modelview matrix to give the

combined transformation matrix, and send that to the shader program. \*/

let modelViewProjection = m4.multiply(projection, matAccum1 );

gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection );

const normal = m4.identity();

m4.inverse(modelView, normal);

m4.transpose(normal, normal);

gl.uniformMatrix4fv(shProgram.iNormalMatrix, false, normal);

/\* Draw the six faces of a cube, with different colors. \*/

gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);

let a = hexToRgb(document.getElementById('ambient').value)

gl.uniform3fv(shProgram.iAmbientColor, [a.r, a.g, a.b]);

let d = hexToRgb(document.getElementById('diffuse').value)

gl.uniform3fv(shProgram.iDiffuseColor, [d.r, d.g, d.b]);

let s = hexToRgb(document.getElementById('specular').value)

gl.uniform3fv(shProgram.iSpecularColor, [s.r, s.g, s.b]);

gl.uniform3fv(shProgram.iLightPosition, [0.75 \* Math.sin(Date.now() \* 0.001), 0.25 \* Math.cos(Date.now() \* 0.001), 0.5]);

gl.uniform3fv(shProgram.iRelativePosition, parabolicHummingTop(textureRelative[0], textureRelative[1]));

gl.uniform2fv(shProgram.iTextureRelative, [map(textureRelative[0], -h, h, 0, 1), map(textureRelative[1], 0, 2 \* Math.PI, 0, 1)]);

gl.uniform1f(shProgram.iAngleOfRotation, parseFloat(document.getElementById('angleOfRotation').value));

surface.Draw();

gl.uniform1i(shProgram.iTranslateLight, true);

lightVis.Draw();

gl.uniform1i(shProgram.iTranslateLight, false);

gl.uniform1i(shProgram.iTranslateRelative, true);

relativeVis.Draw();

gl.uniform1i(shProgram.iTranslateRelative, false);

}