**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 2**

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Студент: Ильин Илья Олегович

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и

масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

Вариант 3. Параллелепипед

1. Описание программы

ЛР написана на языке Swift в виде приложения на iPhone, использовался framework Metal, который по техническим характеристикам является аналогом Open GL, однако Metal более оптимизирован для использования на мобильных и других устройствах компании Apple.

Класс Uniform Buffer используется для повторного использования вычисленных матриц.

Структура Vertex применяется для хранения данных вершины: координаты и цвета.

Класс Node является дочерним для любых фигур, инкапсулирует в себя функции настройки отрисовки (передаются ему при создании), саму отрисовку с построением всех примитивов. Класс Parallelepiped наследует данную функциональность, при этом объявляя конкретные точки параллелепипеда.

В Shaders.metal содержаться шейдеры.

Класс MetalViewController отвечает за создание всех объектов (MTLDevice, projectionMatrix и тд), необходимых делегатов и настройки projectionMatrix. Класс ViewController наследует от данного класса, добавляя функционал жестов и отвечает за конечное воспроизведение картинки на экране устройства.

1. Результаты выполнения тестов

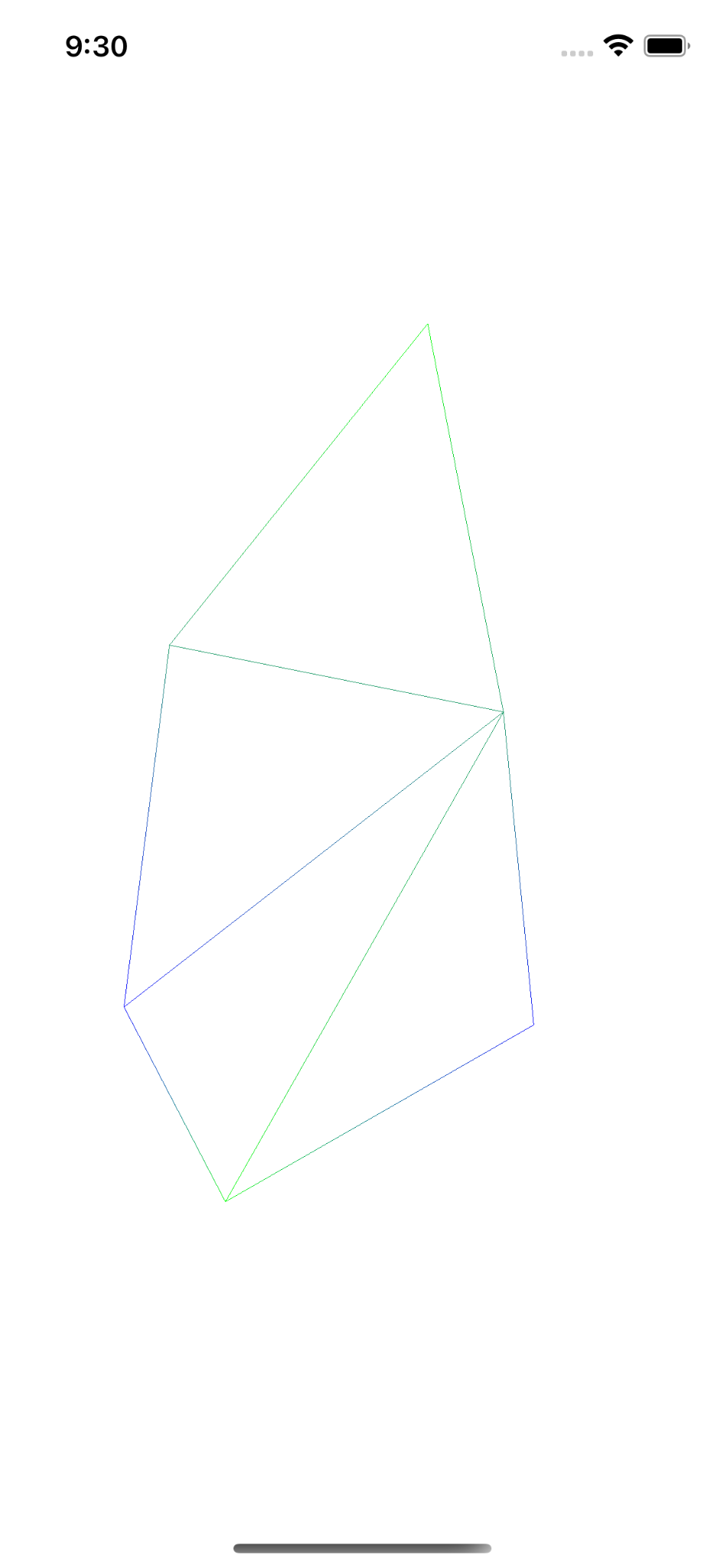
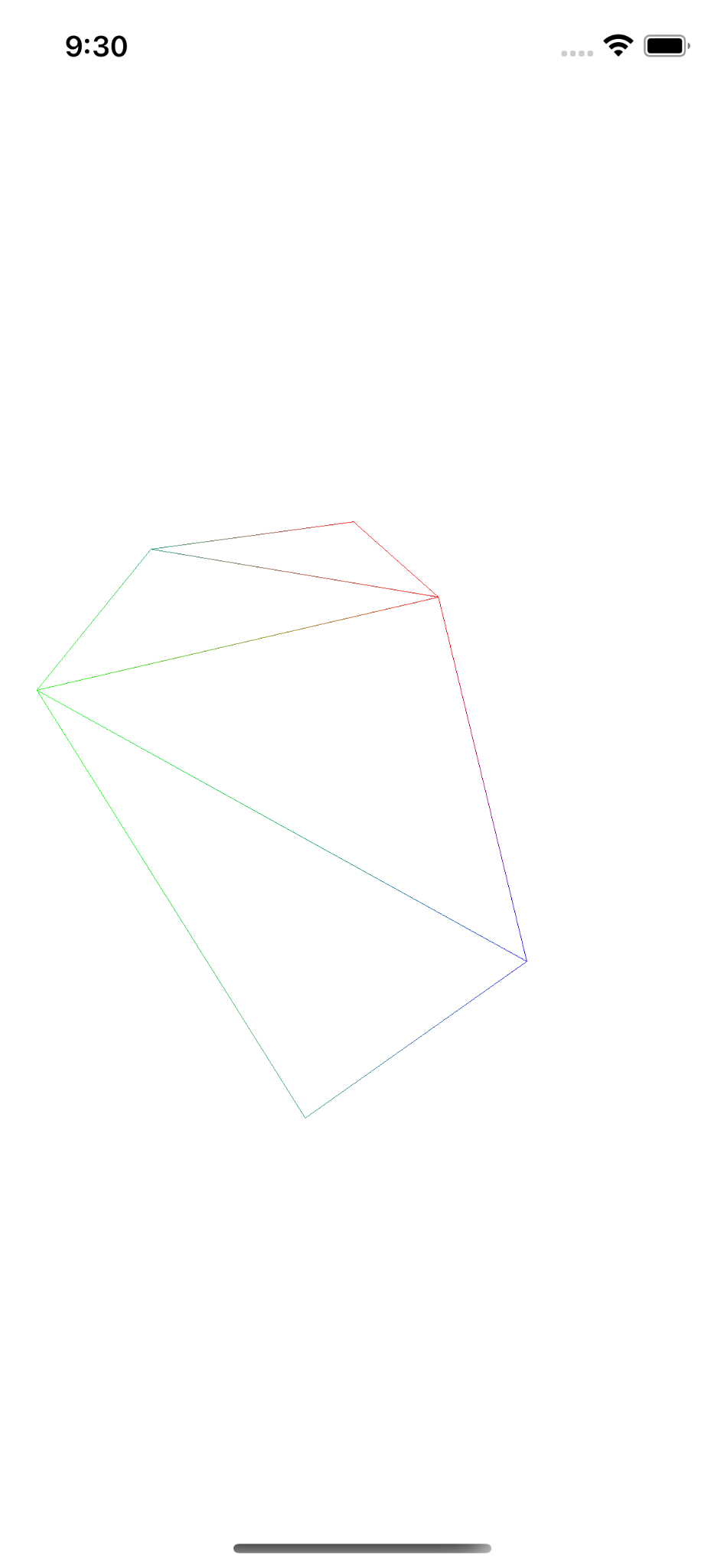
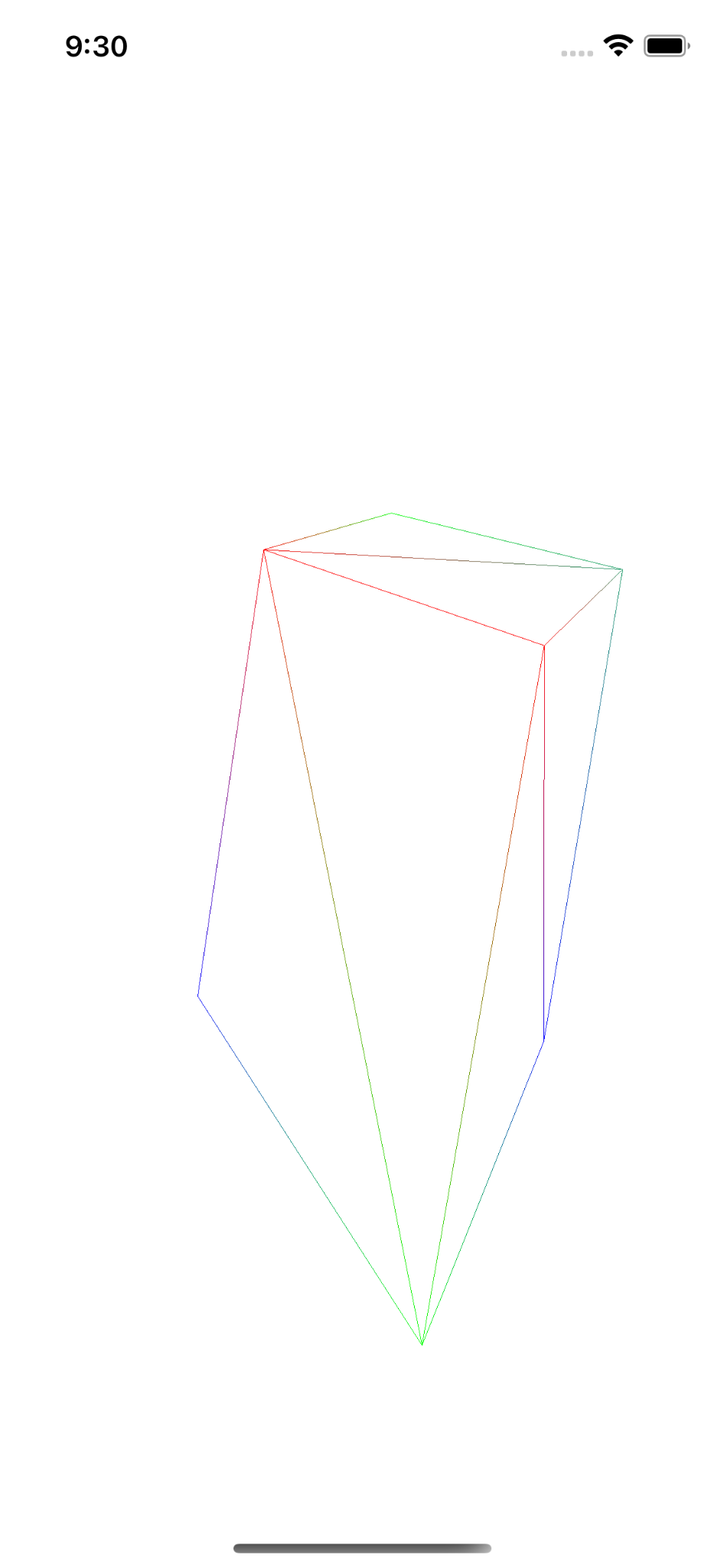
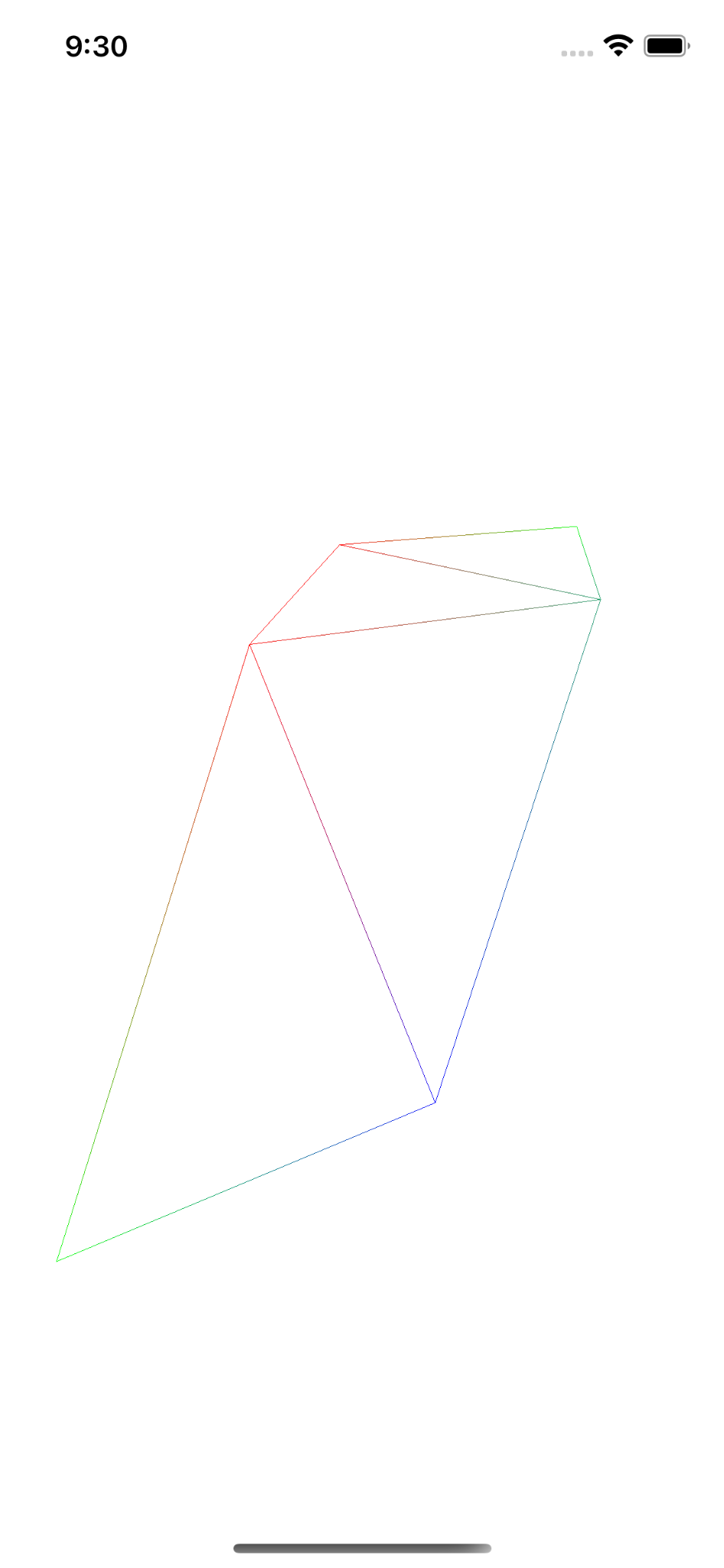


Рис. параллелепипед в различных проекциях

1. Листинг программы

Shaders.metal

//

// Shaders.metal

// CG2

//

// Created by Илья Ильин on 18.10.2021.

//

#include <metal\_stdlib>

**using** **namespace** metal;

**struct** VertexIn{

packed\_float3 position;

packed\_float4 color;

};

**struct** VertexOut{

float4 position [[position]];

float4 color;

};

**struct** Uniforms{

float4x4 modelMatrix;

float4x4 projectionMatrix;

};

**vertex** VertexOut basic\_vertex(**const** **device** VertexIn\* vertex\_array [[ buffer(0) ]],

**const** **device** Uniforms& uniforms [[ buffer(1) ]],

**unsigned** **int** vid [[ vertex\_id ]])

{

float4x4 mv\_Matrix = uniforms.modelMatrix;

float4x4 proj\_Matrix = uniforms.projectionMatrix;

VertexIn VertexIn = vertex\_array[vid];

VertexOut VertexOut;

VertexOut.position = proj\_Matrix \* mv\_Matrix \* float4(VertexIn.position,1);

VertexOut.color = VertexIn.color;

**return** VertexOut;

}

**fragment** half4 basic\_fragment(VertexOut interpolated [[stage\_in]]) {

**return** half4(interpolated.color[0], interpolated.color[1], interpolated.color[2], interpolated.color[3]);

}

BufferProvider.swift

//

// BufferProvider.swift

// CG2

//

// Created by Илья Ильин on 18.10.2021.

//

**import** Foundation

**import** Metal

**import** simd

/// Helper class for reusing Uniform Buffers

**class** BufferProvider: NSObject {

**let** inflightBuffersCount: Int

**private** **var** uniformsBuffers: [MTLBuffer]

**private** **var** avaliableBufferIndex: Int = 0

**var** avaliableResourcesSemaphore: DispatchSemaphore

/// Creates an object of BufferProvider, that can be used to get available reusable buffer

/// - **Parameters**:

/// - device: metal device to use

/// - inflightBuffersCount: number of buffers stored by BufferProvider

/// - sizeOfUniformsBuffer: size of buffer in bytes

**init**(device: MTLDevice, inflightBuffersCount: Int, sizeOfUniformsBuffer: Int) {

**self**.avaliableResourcesSemaphore = DispatchSemaphore(value: inflightBuffersCount)

**self**.inflightBuffersCount = inflightBuffersCount

uniformsBuffers = [MTLBuffer]()

**for** \_ **in** 0...inflightBuffersCount-1 {

**guard** **let** uniformsBuffer = device.makeBuffer(length: sizeOfUniformsBuffer, options: []) **else** {

fatalError("Could not make buffer")

}

uniformsBuffers.append(uniformsBuffer)

}

}

**deinit**{

**for** \_ **in** 0...**self**.inflightBuffersCount{

**self**.avaliableResourcesSemaphore.signal()

}

}

/// Fetching next available buffer and copy data into it

/// - **Parameters**:

/// - Returns: next available uniform buffer

**func** nextUniformsBuffer(projectionMatrix: float4x4, modelViewMatrix: float4x4) -> MTLBuffer {

**let** buffer = uniformsBuffers[avaliableBufferIndex]

**let** bufferPointer = buffer.contents()

// 1

**var** projectionMatrix = projectionMatrix

**var** modelViewMatrix = modelViewMatrix

// 2

memcpy(bufferPointer,

&modelViewMatrix,

MemoryLayout<Float>.size \* float4x4.numberOfElements()

)

memcpy(bufferPointer + MemoryLayout<Float>.size\*float4x4.numberOfElements(),

&projectionMatrix,

MemoryLayout<Float>.size \* float4x4.numberOfElements()

)

avaliableBufferIndex += 1

**if** avaliableBufferIndex == inflightBuffersCount{

avaliableBufferIndex = 0

}

**return** buffer

}

}

Vertex.swift

//

// Vertex.swift

// CG2

//

// Created by Илья Ильин on 18.10.2021.

//

**import** Foundation

**struct** Vertex{

**var** x,y,z: Float // position data

**var** r,g,b,a: Float // color data

**init**(x: Float, y: Float, z: Float, r: Float, g: Float, b: Float, a: Float) {

**self**.x = x

**self**.y = y

**self**.z = z

**self**.r = r

**self**.g = g

**self**.b = b

**self**.a = a

}

**init**(x: Float, y: Float, z: Float) {

**self**.**init**(x: x, y: y, z: z, r: 0, g: 0, b: 0, a: 1)

}

**func** floatBuffer() -> [Float] {

**return** [x,y,z,r,g,b,a]

}

}

Node.swift

//

// Node.swift

// CG2

//

// Created by Илья Ильин on 18.10.2021.

//

**import** Foundation

**import** QuartzCore

**import** Metal

**import** simd

**class** Node {

**let** device: MTLDevice

**let** name: String

**var** vertexCount: Int

**var** vertexBuffer: MTLBuffer

**var** time:CFTimeInterval = 0.0

**var** bufferProvider: BufferProvider

**var** positionX: Float = 0.0

**var** positionY: Float = 0.0

**var** positionZ: Float = 0.0

**var** rotationX: Float = 0.0

**var** rotationY: Float = 0.0

**var** rotationZ: Float = 0.0

**var** scale: Float = 1.0

**init**(name: String, vertices: Array<Vertex>, device: MTLDevice){

**var** vertexData = Array<Float>()

**for** vertex **in** vertices{

vertexData += vertex.floatBuffer()

}

**let** dataSize = vertexData.count \* MemoryLayout.size(ofValue: vertexData[0])

vertexBuffer = device.makeBuffer(bytes: vertexData, length: dataSize, options: [])!

**self**.name = name

**self**.device = device

vertexCount = vertices.count

**self**.bufferProvider = BufferProvider(device: device, inflightBuffersCount: 3, sizeOfUniformsBuffer: MemoryLayout<Float>.size \* float4x4.numberOfElements() \* 2)

}

**func** render(commandQueue: MTLCommandQueue,

pipelineState: MTLRenderPipelineState,

drawable: CAMetalDrawable,

parentModelViewMatrix: float4x4,

projectionMatrix: float4x4,

clearColor: MTLClearColor?)

{

\_ = bufferProvider.avaliableResourcesSemaphore.wait(timeout: DispatchTime.distantFuture)

**let** renderPassDescriptor = MTLRenderPassDescriptor()

renderPassDescriptor.colorAttachments[0].texture = drawable.texture

renderPassDescriptor.colorAttachments[0].loadAction = .clear

**if** **let** clearColor = clearColor {

renderPassDescriptor.colorAttachments[0].clearColor = clearColor

} **else** {

renderPassDescriptor.colorAttachments[0].clearColor = MTLClearColor(red: 255, green: 255, blue: 255, alpha: 1.0)

}

renderPassDescriptor.colorAttachments[0].storeAction = .store

**let** commandBuffer = commandQueue.makeCommandBuffer()!

/\* When the GPU finishes rendering, it executes a completion handler to signal the semaphore and bumps its count back up again. \*/

commandBuffer.addCompletedHandler { (\_) **in**

**self**.bufferProvider.avaliableResourcesSemaphore.signal()

}

**let** renderEncoder = commandBuffer.makeRenderCommandEncoder(descriptor: renderPassDescriptor)!

//For now cull mode is used instead of depth buffer

renderEncoder.setCullMode(MTLCullMode.front)

renderEncoder.setRenderPipelineState(pipelineState)

renderEncoder.setVertexBuffer(vertexBuffer, offset: 0, index: 0)

renderEncoder.setTriangleFillMode(.lines)

**var** nodeModelMatrix = **self**.modelMatrix()

nodeModelMatrix.multiplyLeft(parentModelViewMatrix)

**let** uniformBuffer = bufferProvider.nextUniformsBuffer(projectionMatrix: projectionMatrix, modelViewMatrix: nodeModelMatrix)

renderEncoder.setVertexBuffer(uniformBuffer, offset: 0, index: 1)

renderEncoder.drawPrimitives(type: .triangle, vertexStart: 0, vertexCount: vertexCount, instanceCount: vertexCount/3)

renderEncoder.endEncoding()

commandBuffer.present(drawable)

commandBuffer.commit()

}

**func** modelMatrix() -> float4x4 {

**var** matrix = float4x4()

matrix.translate(positionX, y: positionY, z: positionZ)

matrix.rotateAroundX(rotationX, y: rotationY, z: rotationZ)

matrix.scale(scale, y: scale, z: scale)

**return** matrix

}

**func** updateWithDelta(delta: CFTimeInterval){

time += delta

}

}

Parallelepiped.swift

//

// Cube.swift

// CG2

//

// Created by Илья Ильин on 18.10.2021.

//

**import** Foundation

**import** Metal

**class** Parallelepiped: Node {

**init**(device: MTLDevice) {

**let** A = Vertex(x: -0.9, y: 1.0, z: 1.0, r: 1.0, g: 0.0, b: 0.0, a: 1.0)

**let** B = Vertex(x: -1.7, y: -1.0, z: 1.0, r: 0.0, g: 1.0, b: 0.0, a: 1.0)

**let** C = Vertex(x: -0.2, y: -1.0, z: 1.0, r: 0.0, g: 0.0, b: 1.0, a: 1.0)

**let** D = Vertex(x: 0.4, y: 1.0, z: 1.0, r: 0.1, g: 0.6, b: 0.4, a: 1.0)

**let** Q = Vertex(x: 0.2, y: 1.0, z: -0.6, r: 1.0, g: 0.0, b: 0.0, a: 1.0)

**let** R = Vertex(x: 1.7, y: 1.0, z: -0.6, r: 0.0, g: 1.0, b: 0.0, a: 1.0)

**let** S = Vertex(x: -0.6, y: -1.0, z: -0.6, r: 0.0, g: 0.0, b: 1.0, a: 1.0)

**let** T = Vertex(x: 0.9, y: -1.0, z: -0.6, r: 0.1, g: 0.6, b: 0.4, a: 1.0)

**let** verticesArray:Array<Vertex> = [

A,B,C ,A,C,D, //Front

R,T,S ,Q,R,S, //Back

Q,S,B ,Q,B,A, //Left

D,C,T ,D,T,R, //Right

Q,A,D ,Q,D,R, //Top

B,S,T ,B,T,C //Bot

]

**super**.**init**(name: "Cube", vertices: verticesArray, device: device)

}

}

1. Литература
2. Документация Swift [Электронный ресурс] URL: https://swift.org/documentation/ (Дата обращения: 20.09.2021).