**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения фотореалистичных изображений

Студент: Ильин Илья Олегович

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант многогранника: 14. Усеченный прямой эллиптический цилиндр.

1. Описание программы

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 TiltCylinder наследует данную функциональность, при этом вычисляя конкретные точки цилиндра (для вычисления точек эллипсов используется алгоритм Брезенхэма).

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. Набор тестов

1. Изначальный вид.

2. Вращение многогранника + незначительное увеличение точности аппроксимации .

3. Вид сбоку, средняя точность апроксимации

4. Максимальная точность апроксимации, вид с разных сторон

1. Результаты выполнения тестов

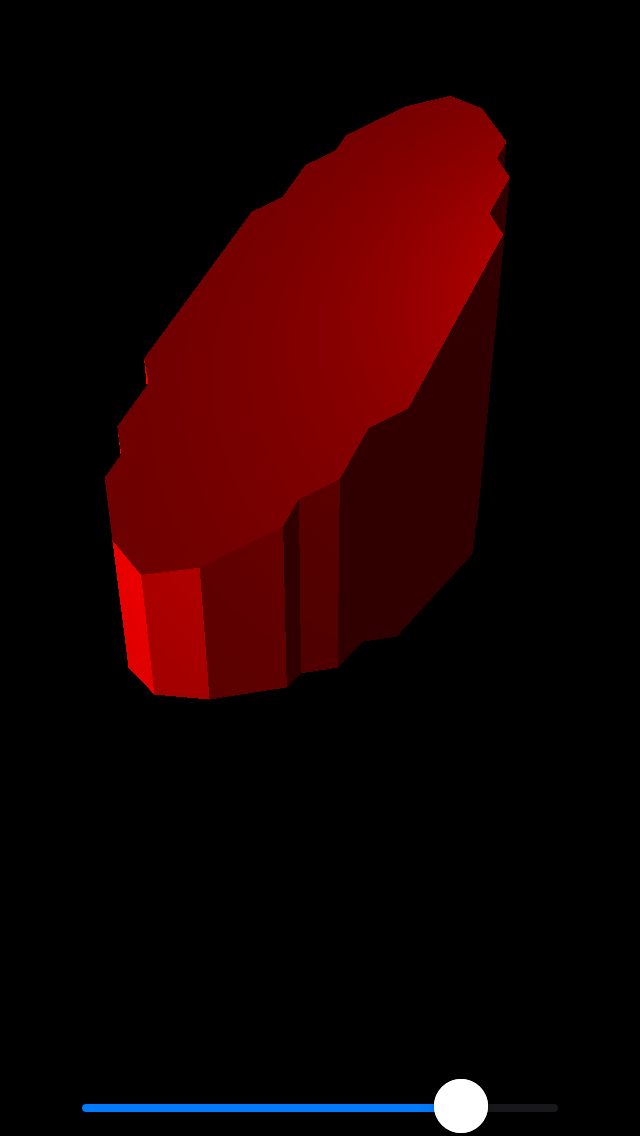
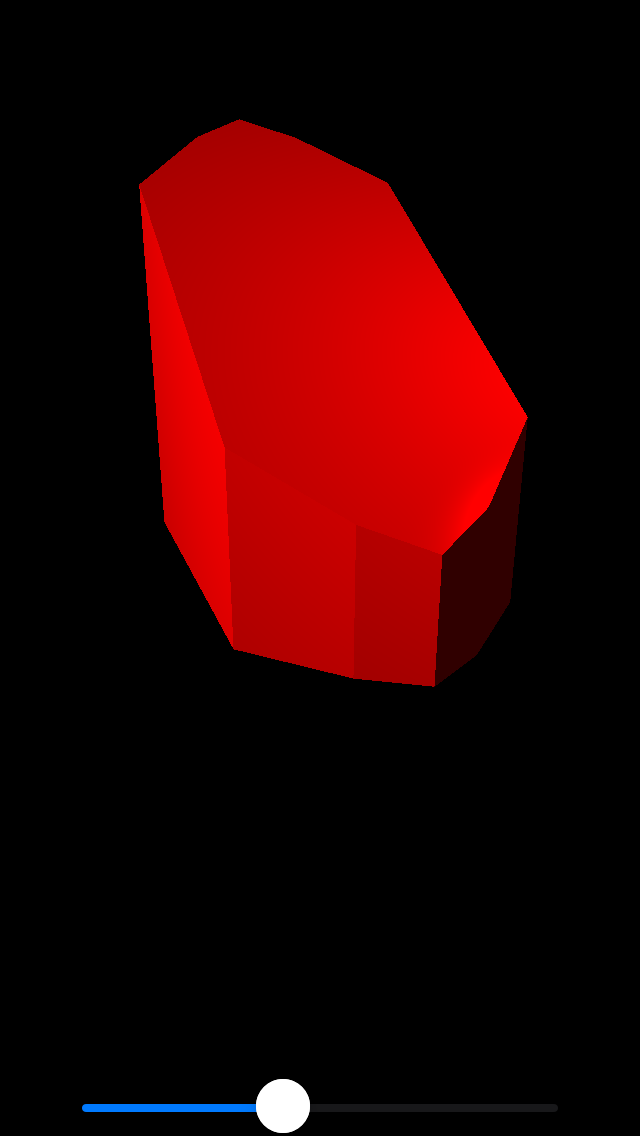
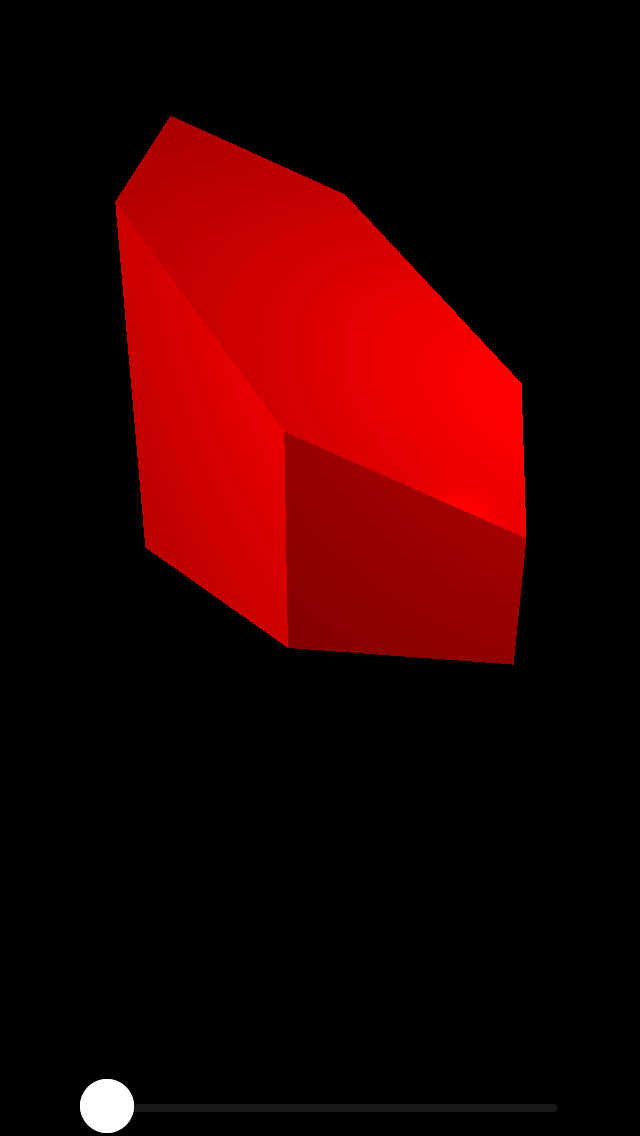


Рис. 4.1 Изначальный вид.

Рис. 4.2 Вращение многогранника + незначительное увеличение точности аппроксимации .

Рис. 4.3 Вид сбоку, средняя точность апроксимации

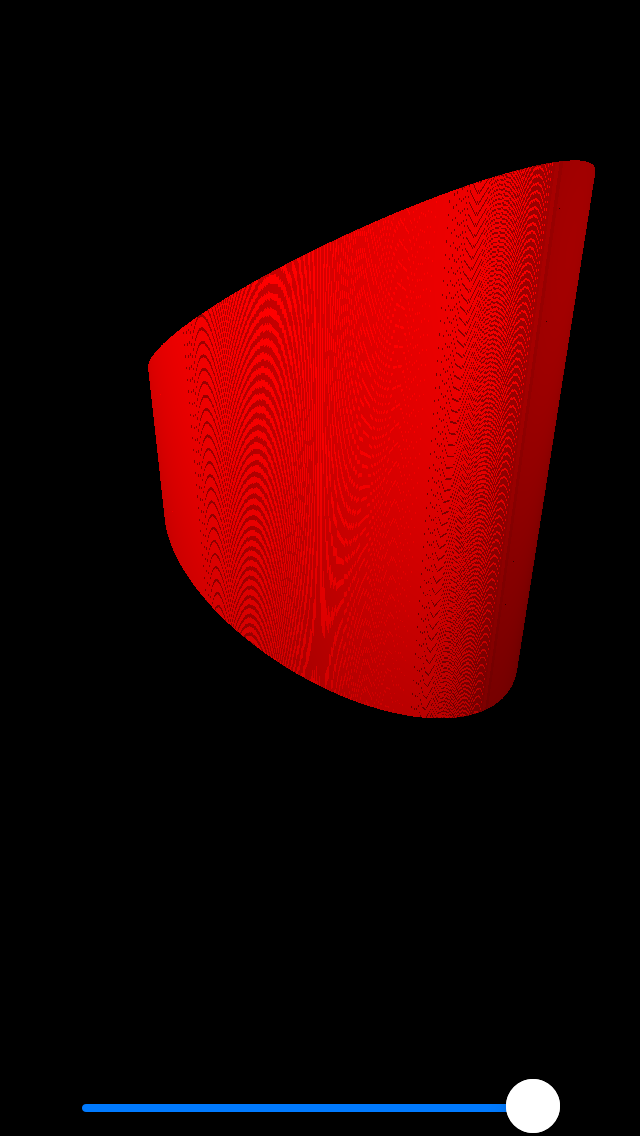
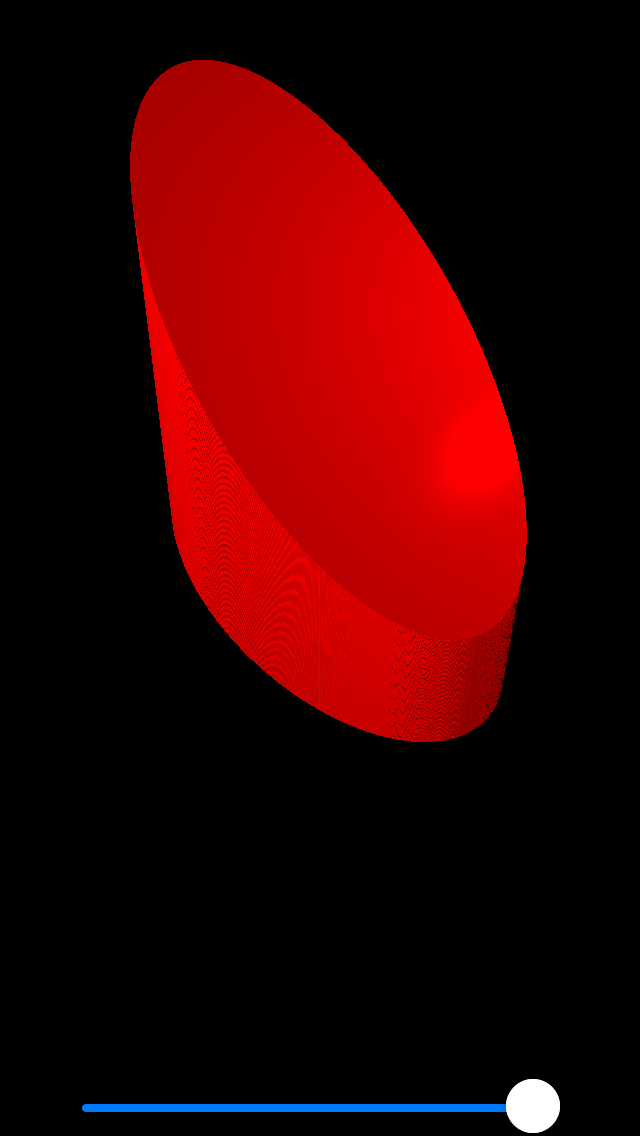


Рис. 4.4. Максимальная точность апроксимации, вид с разных сторон

1. Листинг программы

TiltCylinder.swift

**import** Foundation

**import** Metal

**import** simd

**import** UIKit

**class** TiltCylinder: Node {

**private** **var** color: (Float, Float, Float) = (0.5, 0.5, 0.5)

**init**(device: MTLDevice){

**let** a: Float = 2

**let** b: Float = 1

**let** delta: Float = 0.01

**let** ellipseVertices = TiltCylinder.countEllipseVertices(a: a, b: b, delta: delta)

ellipseVertices.forEach { print("(\($0.x), \($0.y))") }

/// bottom ellipse

**let** bottomEllipseMatrix = makeRotationMatrixAroundX(degrees: 90)

**let** bottomEllipse = ellipseVertices.map { bottomEllipseMatrix \* $0 }

**let** bottomCenter = bottomEllipseMatrix \* simd\_float4(x: 0, y: 0, z: 0, w: 1)

/// upped ellipse

**let** upperEllipseMatrix = makeTranslationMatrix(tx: 0, ty: 2, tz: 0) \* makeRotationMatrixAroundZ(degrees: 30) \* makeRotationMatrixAroundX(degrees: 90)

**let** upperEllipse = ellipseVertices.map { upperEllipseMatrix \* $0 }

**let** upperCenter = upperEllipseMatrix \* simd\_float4(x: 0, y: 0, z: 0, w: 1)

/// count mesh

**var** verticesArray = [Vertex]()

verticesArray += TiltCylinder.makeEllipseMesh(ellipseVectors: bottomEllipse.reversed(), center: bottomCenter, color: color)

verticesArray += TiltCylinder.makeEllipseMesh(ellipseVectors: upperEllipse, center: upperCenter, color: color)

verticesArray += TiltCylinder.makeSideSurfaceMesh(upperEllipse: upperEllipse.reversed(), bottomEllipse: bottomEllipse.reversed(), color: color)

// debug log

print("total \(verticesArray.count) vertices as \(verticesArray.count / 3) triangles")

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

}

**class** **func** makeSideSurfaceMesh(upperEllipse: [simd\_float4], bottomEllipse: [simd\_float4], color: (Float, Float, Float)) -> [Vertex] {

**if** upperEllipse.isEmpty || bottomEllipse.isEmpty {

**return** []

}

**let** circledUpperEllipse = upperEllipse + [upperEllipse.first!]

**let** circledBottomEllipse = bottomEllipse + [bottomEllipse.first!]

**var** vertices = [Vertex]()

**var** prevVectors: (simd\_float4, simd\_float4)? = **nil**

**for** (upVector, bottomVector) **in** zip(circledUpperEllipse, circledBottomEllipse) {

**if** **let** prevVectors = prevVectors {

/// divide in 2 triangles

**let** prevUpVector = prevVectors.0

**let** prevBottomVector = prevVectors.1

**let** normal1 = countNormal(v1: prevBottomVector, v2: prevUpVector, v3: upVector)

**let** normal2 = countNormal(v1: prevBottomVector, v2: upVector, v3: bottomVector)

**let** v11 = Vertex(vector: prevBottomVector, color: color, normal: normal1)

**let** v12 = Vertex(vector: prevUpVector, color: color, normal: normal1)

**let** v13 = Vertex(vector: upVector, color: color, normal: normal1)

**let** v21 = Vertex(vector: prevBottomVector, color: color, normal: normal2)

**let** v22 = Vertex(vector: upVector, color: color, normal: normal2)

**let** v23 = Vertex(vector: bottomVector, color: color, normal: normal2)

vertices += [v11, v12, v13]

vertices += [v21, v22, v23]

}

prevVectors = (upVector, bottomVector)

}

**return** vertices

}

**class** **func** makeEllipseMesh(ellipseVectors: [simd\_float4], center: simd\_float4, color: (Float, Float, Float)) -> [Vertex] {

**var** vertices = [Vertex]()

**if** !ellipseVectors.isEmpty {

**var** prevVector: simd\_float4? = **nil**

**for** vector **in** ellipseVectors + [ellipseVectors.first!] {

**if** **let** prevVector = prevVector {

/// as triangles

**let** normal = countNormal(v1: center, v2: prevVector, v3: vector)

**let** prevVertex = Vertex(vector: prevVector, color: color, normal: normal)

**let** vertex = Vertex(vector: vector, color: color, normal: normal)

**let** centerVertex = Vertex(vector: center, color: color, normal: normal)

vertices.append(centerVertex)

vertices.append(prevVertex)

vertices.append(vertex)

}

prevVector = vector

}

}

**return** vertices

}

/// Count Ellipse points, located in Oxz plane

/// - **Parameters**:

/// - a: semi-major axis

/// - b: semi-minor axis

/// - delta: approximation accuracy

/// - color: color of ellipse

/// - **Returns**: array of ellipse vertexes

**class** **func** countEllipseVertices(a: Float, b: Float, delta: Float) -> [simd\_float4] {

**let** a2: Float = a \* a

**let** b2: Float = b \* b

**var** x: Float = 0

**var** y: Float = b

**func** f(x: Float, y: Float) -> Float {

**return** b2\*x\*x + a2\*y\*y - a2\*b2

}

**var** vertices = [simd\_float4]()

/// upper arc

**while** (a2 \* (y - delta/2)) > (b2 \* (x+delta)) {

**let** vertex = simd\_float4(x: x, y: y, z: 0, w: 1)

vertices.append(vertex)

/// choose next point between (x + delta, y) and (x + delta, y - delta)

**if** (f(x: x + delta, y: y - delta/2) >= 0) {

y -= delta

}

x += delta

}

/// bottom arc

**while** y >= 0 {

**let** vertex = simd\_float4(x: x, y: y, z: 0, w: 1)

vertices.append(vertex)

/// choose between (x + delta, y - delta) and (x, y - delta)

**if** (f(x: x + delta/2, y: y - delta) <= 0) {

x += delta

}

y -= delta

}

**if** **let** prevVertex = vertices.last {

**let** endVertex = simd\_float4(x: a, y: 0, z: 0, w: 1)

**if** endVertex != prevVertex {

vertices.append(endVertex)

}

}

/// plot other arcs using symmetry

**let** arc2: [simd\_float4]

**let** arc3: [simd\_float4]

**let** arc4: [simd\_float4]

**if** vertices.count > 2 {

arc2 = vertices[1..<vertices.count-1].map { (v) -> simd\_float4 **in**

**var** newV = v

newV.x = v.x \* -1

**return** newV

}.reversed()

} **else** {

arc2 = []

}

**if** vertices.count > 1 {

arc3 = vertices[1...].map { (v) -> simd\_float4 **in**

**var** newV = v

newV.x = v.x \* -1

newV.y = v.y \* -1

**return** newV

}

arc4 = vertices[0..<vertices.count-1].map { (v) -> simd\_float4 **in**

**var** newV = v

newV.y = v.y \* -1

**return** newV

}.reversed()

} **else** {

arc3 = []

arc4 = []

}

vertices += arc4 + arc3 + arc2

**return** vertices

}

}

ViewController.swift

# **ЛР 3**

## **Вариант 14. Усеченный прямой эллиптический цилиндр.**

Используя результаты Л.Р.No2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

\*/

**import** UIKit

**import** simd

**class** ViewController: MetalViewController, MetalViewControllerDelegate {

**var** worldModelMatrix: float4x4!

**var** objectToDraw: TiltCylinder!

**@IBOutlet** **weak** **var** slider: UISlider!

**let** minDelta: Float = 0.001

**let** maxDelta: Float = 2

**let** panSensivity: Float = 5.0

**let** pinchSensivity: Float = 0.5

**var** lastPanLocation: CGPoint!

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

worldModelMatrix = float4x4()

worldModelMatrix.translate(0.0, y: 0.0, z: -4)

worldModelMatrix.rotateAroundX(float4x4.degrees(toRad: 25), y: 0.0, z: 0.0)

objectToDraw = TiltCylinder(device: device)

**self**.metalViewControllerDelegate = **self**

setupGestures()

}

//**MARK: - MetalViewControllerDelegate**

**func** renderObjects(drawable: CAMetalDrawable) {

objectToDraw.render(commandQueue: commandQueue, pipelineState: pipelineState, drawable: drawable, parentModelViewMatrix: worldModelMatrix, projectionMatrix: projectionMatrix, clearColor: MTLClearColor(red: 1.0, green: 1.0, blue: 1.0, alpha: 1.0))

}

**func** updateLogic(timeSinceLastUpdate: CFTimeInterval) {

objectToDraw.updateWithDelta(delta: timeSinceLastUpdate)

}

//**MARK: - Gesture related**

**func** setupGestures(){

**let** pan = UIPanGestureRecognizer(target: **self**, action: **#selector**(ViewController.pan(\_:)))

**let** pinch = UIPinchGestureRecognizer(target: **self**, action: **#selector**(ViewController.pinch(\_:)))

**self**.view.addGestureRecognizer(pan)

**self**.view.addGestureRecognizer(pinch)

}

**@objc** **func** pan(\_ sender: UIPanGestureRecognizer) {

**if** sender.state == UIGestureRecognizer.State.changed {

**let** pointInView = sender.location(in: **self**.view)

**let** xDelta = Float((lastPanLocation.x - pointInView.x)/**self**.view.bounds.width) \* panSensivity

**let** yDelta = Float((lastPanLocation.y - pointInView.y)/**self**.view.bounds.height) \* panSensivity

objectToDraw.rotationY -= xDelta

objectToDraw.rotationX -= yDelta

lastPanLocation = pointInView

} **else** **if** sender.state == UIGestureRecognizer.State.began {

lastPanLocation = sender.location(in: **self**.view)

}

}

**@objc** **func** pinch(\_ sender: UIPinchGestureRecognizer) {

**let** newScale = Float(sender.scale) \* pinchSensivity

objectToDraw.scale = newScale

}

**@objc** **private** **func** startZooming(\_ sender: UIPinchGestureRecognizer) {

**let** scaleResult = sender.view?.transform.scaledBy(x: sender.scale, y: sender.scale)

**guard** **let** scale = scaleResult, scale.a > 1, scale.d > 1 **else** { **return** }

sender.view?.transform = scale

sender.scale = 1

}

}

Vertex.swift

//

// Vertex.swift

// CG3-4

//

// Created by Илья Ильин on 30.11.2021.

//

**import** Darwin

**import** simd

**struct** Vertex{

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

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

**var** nX,nY,nZ: Float // normal

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

**self**.x = x

**self**.y = y

**self**.z = z

**self**.r = r

**self**.g = g

**self**.b = b

**self**.a = a

**self**.nX = nX

**self**.nY = nY

**self**.nZ = nZ

}

**init**(vector: simd\_float4, color: (Float, Float, Float), normal: simd\_float3) {

**self**.x = vector.x

**self**.y = vector.y

**self**.z = vector.z

**self**.r = color.0

**self**.g = color.1

**self**.b = color.2

**self**.a = 1.0

**self**.nX = normal.x

**self**.nY = normal.y

**self**.nZ = normal.z

}

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

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

}

**func** apply(transformMatrix: float4x4) -> Vertex {

**var** vector = simd\_float4(x: x, y: y, z: z, w: 1)

vector = transformMatrix \* vector

**var** vertex = **self**

vertex.x = vector.x

vertex.y = vector.y

vertex.z = vector.z

**return** vertex

}

**static** **func** == (lhs: Vertex, rhs: Vertex) -> Bool {

**let** diff1 = fabsf(lhs.x - rhs.x)

**let** diff2 = fabsf(lhs.y - rhs.y)

**let** diff3 = fabsf(lhs.z - rhs.z)

**let** sum1 = fabsf(lhs.x + rhs.x)

**let** sum2 = fabsf(lhs.y + rhs.y)

**let** sum3 = fabsf(lhs.z + rhs.z)

**let** K: Float = pow(10.0, 3.0)

**return** ( (diff1 < K \* Float.ulpOfOne \* sum1) || diff1 < Float.leastNormalMagnitude ) &&

( (diff2 < K \* Float.ulpOfOne \* sum2) || diff2 < Float.leastNormalMagnitude ) &&

( (diff3 < K \* Float.ulpOfOne \* sum3) || diff3 < Float.leastNormalMagnitude )

}

**static** **func** != (lhs: Vertex, rhs: Vertex) -> Bool {

**return** !(lhs == rhs)

}

}

1. Выводы

В ходе выполнения данной ЛР использовал шаблон, написанный для ЛР 2, также закрепил знания по использованию Core Graphics, улучшил графический интерфейс и научился аппроксимировать тело треугольниками

ЛИТЕРАТУРА

1. Алгоритм Брезенхема для эллипса [Электронный ресурс]URL: <http://grafika.me/node/656> (Дата обращения: 18.10.2021).
2. Справка по фреймворку Core Graphics [Электронный ресурс]URL: <https://developer.apple.com/documentation/coregraphics>

(Дата обращения: 17.10.2021).