**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Курсовой проект**

**по дисциплине «Компьютерная графика»**

Студент: Ильин Илья Олегович

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

**Вариант 15:** Поверхность вращения. Образующая – кубическая кривая Безье 3D

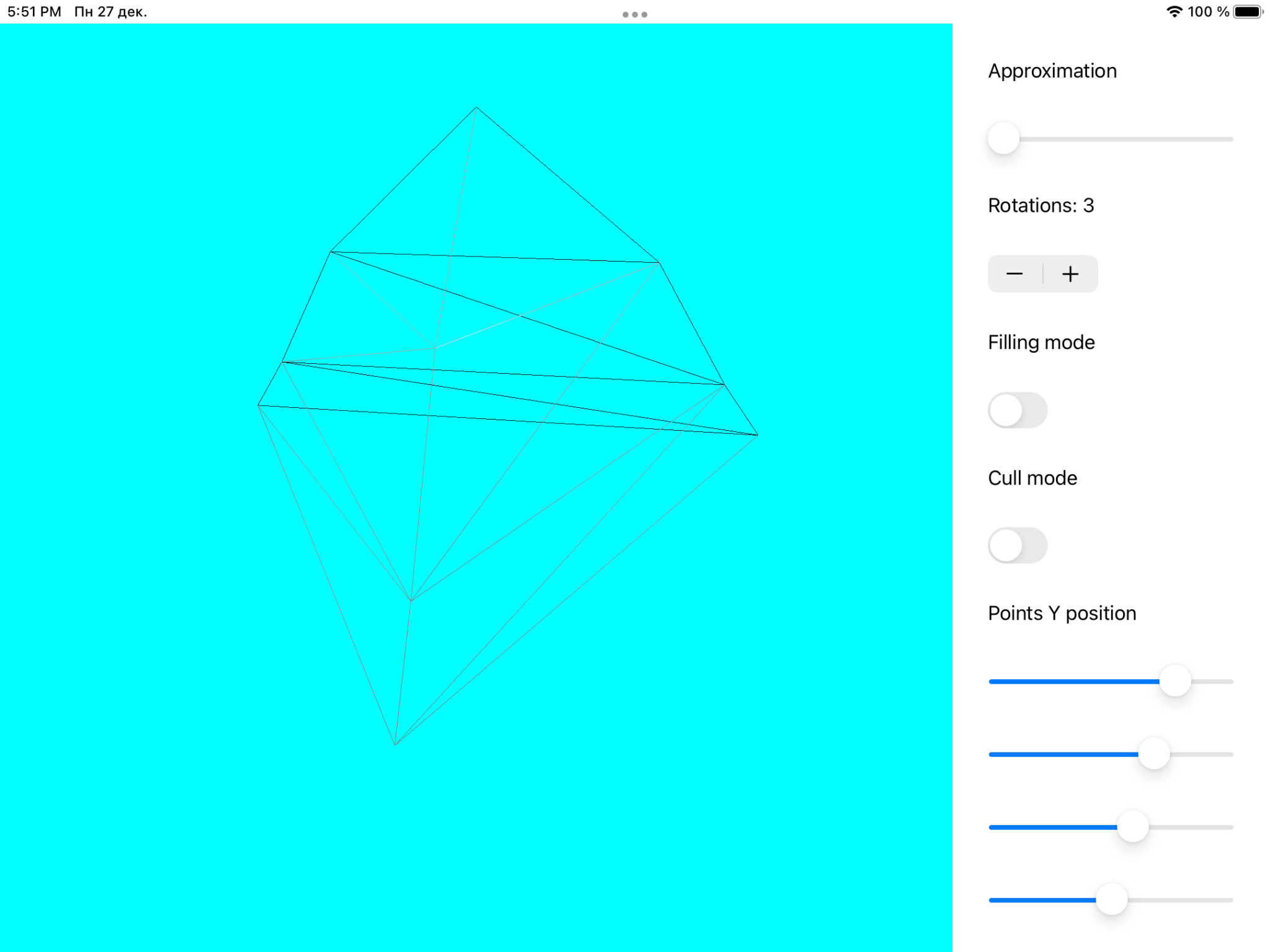
1. Описание программы

Программа написана на Swift, фреймворки Metal и Model I/O. Используются классы и структуры из ЛР6.

Для расчета заданной поверхности был написан класс *RotationBezierCurve*. Сначала считаем кривую Безье по контрольным точкам, затем вращаем данную кривую на заданный угол, параллельно создаем меш из треугольников между прошлой кривой и текущей. Причём, любое изменение количества поворот или положение контрольных точек ставит специальный флаг, затем при очередном цикле отрисовки произойдет пересчет точек поверхности.

Элементы управления позволяют изменять координаты контрольных точек, уровень гладкости кривой Безье, количество вращений, параметры заливки фигуры и удаления невидимых линий.

1. Набор тестов



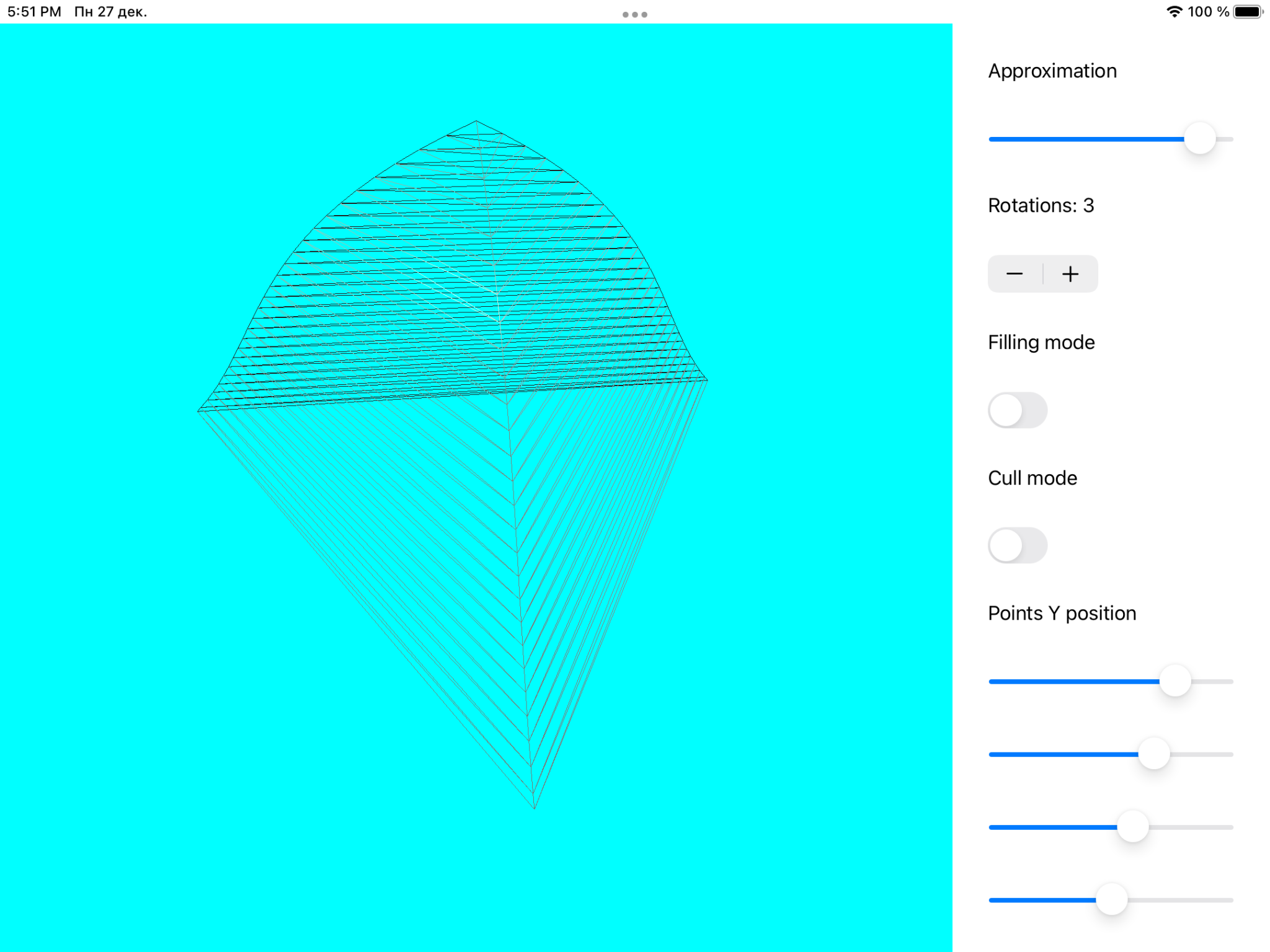
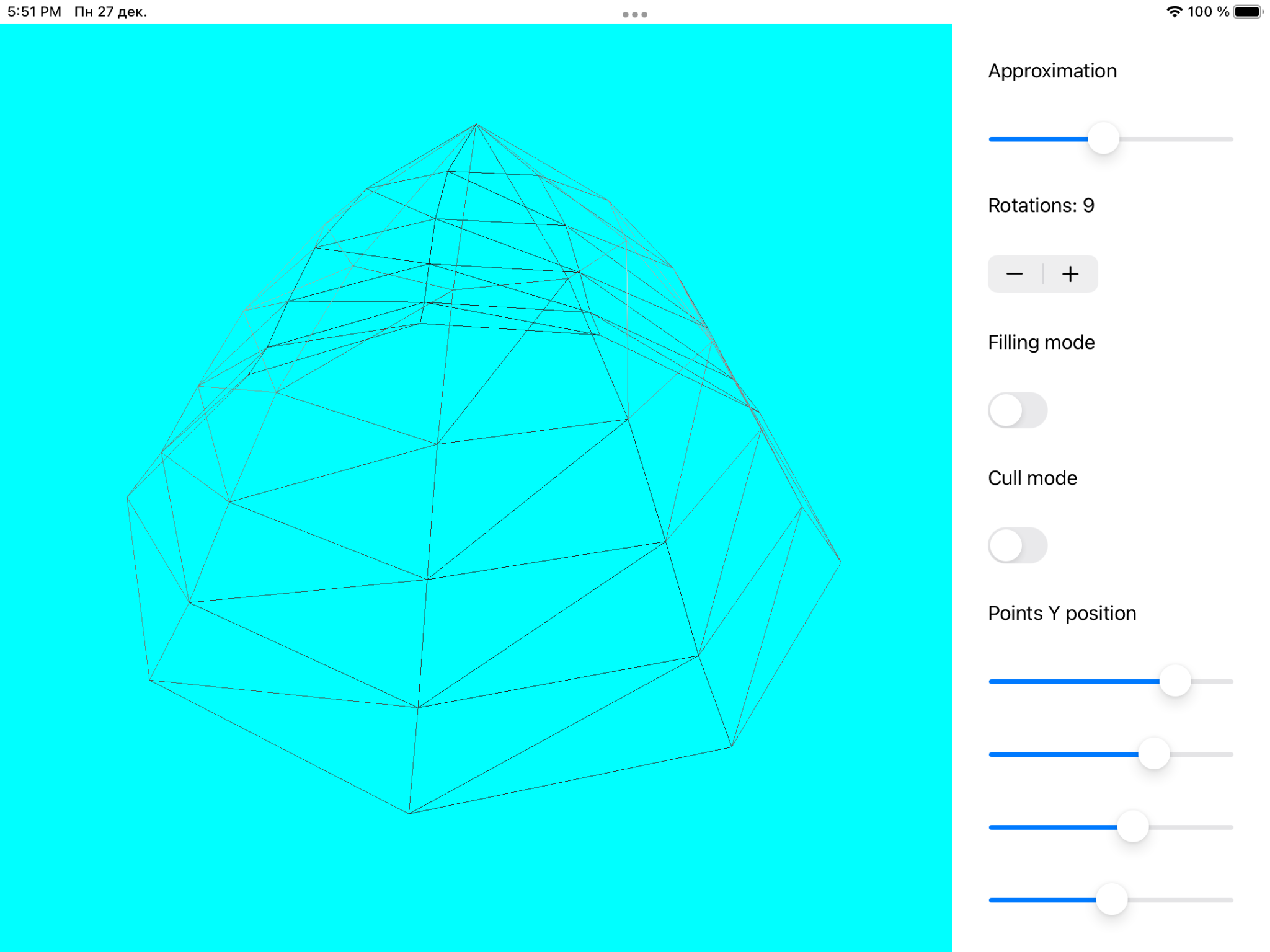


Рис 1. Изначальный вид поверхности с низкой и высокой аппроксимацией



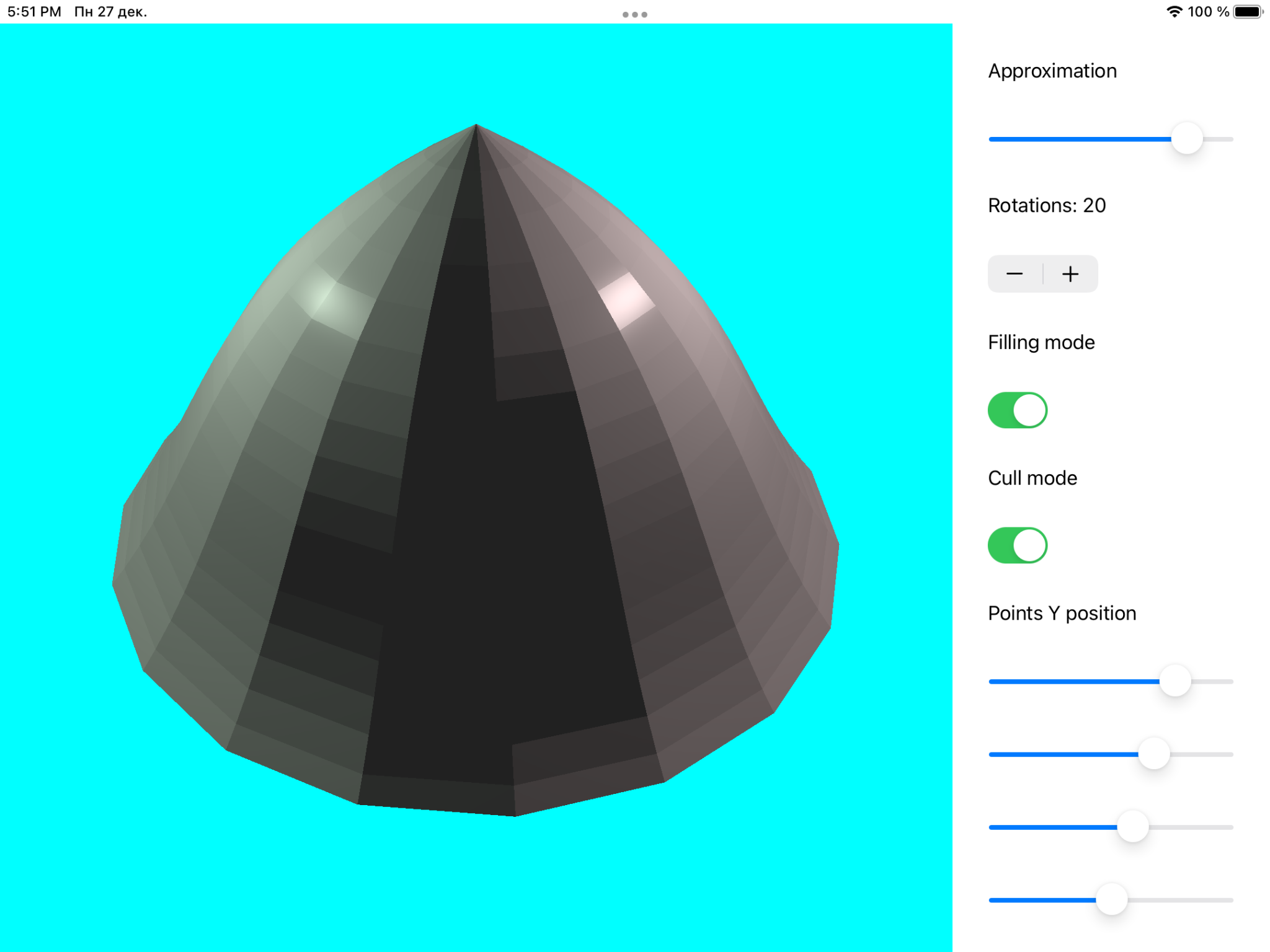


Рис 2. Поверхность со средним числом вращений, вид сетки и текстуры

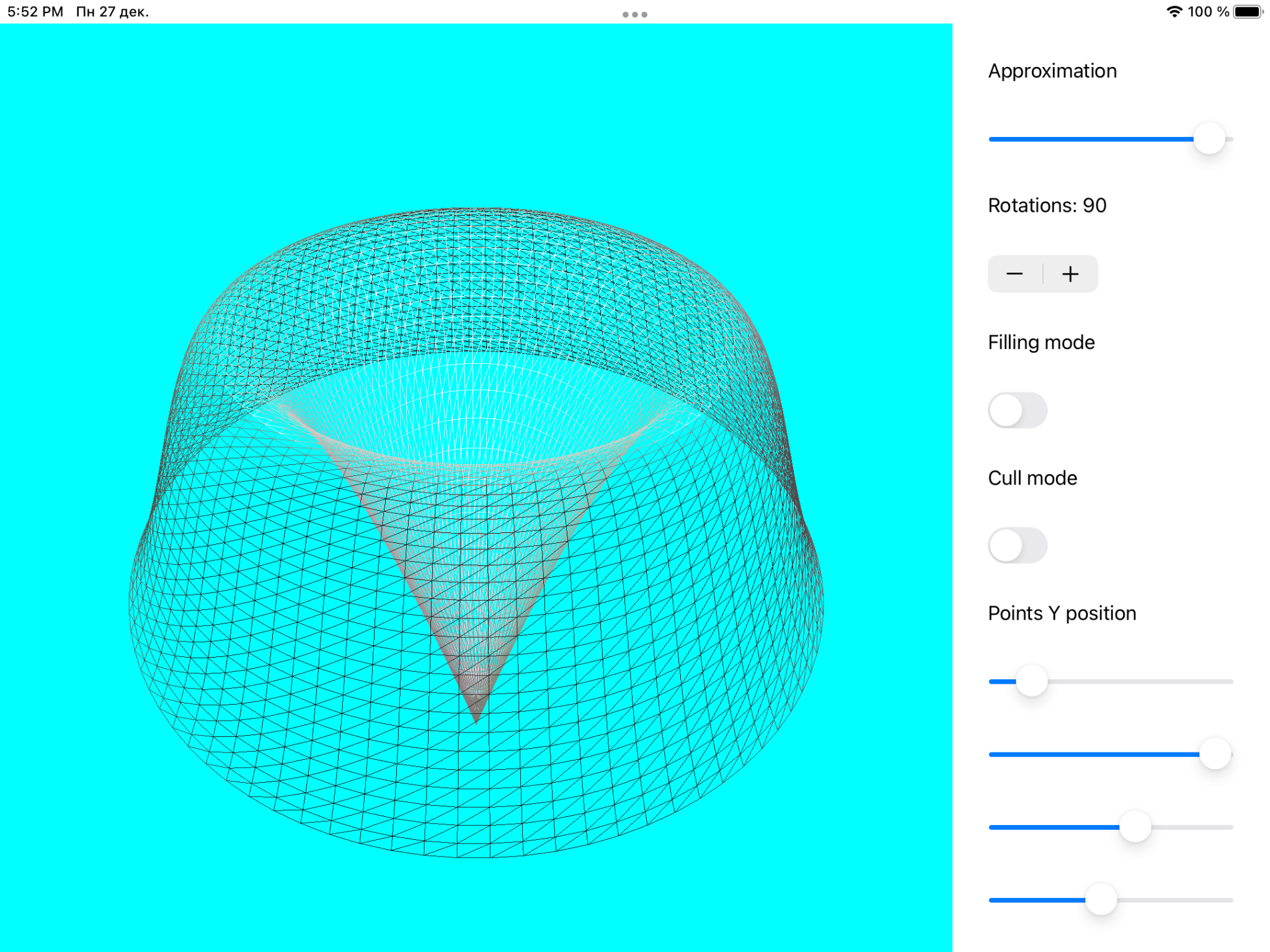


Рис 3. Смещение точек, максимальный уровень аппроксимации и вращений.

1. Листинг программы

ViewController.swift

//

// ViewController.swift

// CG6

//

// Created by Илья Ильин on 16.12.2021.

//

/\*

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

Вариант 15: Поверхность вращения. Образующая – кубическая кривая Безье 3D

\*/

**import** UIKit

**import** MetalKit

**class** ViewController: MetalViewController {

// **MARK: Object related**

**var** object: RotationBezierCurve!

**let** approximationRange: (Float, Float) = (0.4, 0.01)

**var** approximation: Float = 0.4 {

**didSet** {

renderer.approximation = **self**.approximation

}

}

**var** rotation: Int = 3 {

**didSet** {

object.rotations = rotation

}

}

**var** points = [

simd\_float3(0, 3, 0),

simd\_float3(2, 2, 0),

simd\_float3(2, 1, 0),

simd\_float3(3, 0, 0)

] {

**didSet** {

object.controlPoints = points

}

}

**let** rotations = [3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 72, 90]

// **MARK: Views and gestures**

**var** stackView: UIStackView!

**var** deltaSlider: UISlider! /// from 0.001 to 1

**var** pointsSliders: [UISlider]!

**var** rotationStepper: UIStepper!

**var** rotationsLabel: UILabel!

**let** panSensivity: Float = 5.0

**let** pinchSensivity: Float = 0.5

**var** lastPanLocation: CGPoint!

// **MARK: - UIViewController**

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

setupViews()

setupScene()

**super**.viewDidLoad()

setupGestures()

}

// **MARK: - Scene setup**

**func** setupScene() {

**self**.scene = Scene()

**self**.scene.ambientLightColor = simd\_float3(0.01, 0.01, 0.01)

**let** light0 = Light(worldPosition: simd\_float3( 2, 2, 2), color: simd\_float3(1.0, 0.8, 0.8))

**let** light1 = Light(worldPosition: simd\_float3(-2, 2, 2), color: simd\_float3(0.8, 1.0, 0.8))

**let** light2 = Light(worldPosition: simd\_float3( 0, -2, 2), color: simd\_float3(0.8, 0.8, 1.0))

**self**.scene.lights = [ light0, light1, light2 ]

**let** material = Material(specularColor: simd\_float3(0.8, 0.8, 0.8),

specularPower: 100,

color: simd\_float4(0.5, 0.5, 0.5, 1.0),

frontFacing: .counterClockwise,

cullMode: MTLCullMode.none,

triangleFillMode: .lines)

object = RotationBezierCurve(name: "Bezier", material: material, controlPoints: points, approximation: 0.4, rotations: 3)

**self**.scene.rootNode.children.append(object)

}

// **MARK: - Views setup**

**func** setupViews() {

mtkView = MTKView()

mtkView.clearColor = MTLClearColor(red: 0, green: 1, blue: 1, alpha: 1)

mtkView.isOpaque = **false**

mtkView.translatesAutoresizingMaskIntoConstraints = **false**

view.addSubview(mtkView)

stackView = UIStackView()

stackView.translatesAutoresizingMaskIntoConstraints = **false**

stackView.axis = .vertical

stackView.alignment = .fill

stackView.spacing = 20

stackView.distribution = .equalSpacing

**let** deltaLabel = UILabel()

deltaLabel.text = "Approximation"

stackView.addArrangedSubview(deltaLabel)

deltaSlider = UISlider()

deltaSlider.value = 1

deltaSlider.maximumValue = 100

deltaSlider.minimumValue = 1

deltaSlider.addTarget(**self**, action: **#selector**(**self**.deltaValueChanged(\_:)), for: .valueChanged)

stackView.addArrangedSubview(deltaSlider)

rotationsLabel = UILabel()

rotationsLabel.text = "Rotations: \(rotation)"

stackView.addArrangedSubview(rotationsLabel)

rotationStepper = UIStepper()

rotationStepper.wraps = **true**

rotationStepper.minimumValue = 0

rotationStepper.maximumValue = Double(rotations.count - 1)

rotationStepper.value = 0

rotationStepper.addTarget(**self**, action: **#selector**(stepperValueChanged(\_:)), for: .valueChanged)

stackView.addArrangedSubview(rotationStepper)

**let** fillLabel = UILabel()

fillLabel.text = "Filling mode"

stackView.addArrangedSubview(fillLabel)

**let** fillSwitch = UISwitch()

fillSwitch.addTarget(**self**, action: **#selector**(fillValueChanged(\_:)), for: .valueChanged)

fillSwitch.isOn = **false**

stackView.addArrangedSubview(fillSwitch)

**let** cullLabel = UILabel()

cullLabel.text = "Cull mode"

stackView.addArrangedSubview(cullLabel)

**let** cullSwitch = UISwitch()

cullSwitch.addTarget(**self**, action: **#selector**(cullValueChanged(\_:)), for: .valueChanged)

cullSwitch.isOn = **false**

stackView.addArrangedSubview(cullSwitch)

**let** pointsLabel = UILabel()

pointsLabel.text = "Points Y position"

stackView.addArrangedSubview(pointsLabel)

**for** i **in** 0..<points.count {

**let** slider = UISlider()

slider.tag = i

slider.minimumValue = -5

slider.maximumValue = 5

slider.value = points[i].y

slider.addTarget(**self**, action: **#selector**(pointValueChanged(\_:)), for: .valueChanged)

stackView.addArrangedSubview(slider)

}

view.addSubview(stackView)

addConstraints()

}

**func** addConstraints() {

NSLayoutConstraint.activate([

mtkView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor),

mtkView.leftAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leftAnchor),

mtkView.widthAnchor.constraint(equalTo: view.safeAreaLayoutGuide.widthAnchor, multiplier: 0.75),

mtkView.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor),

stackView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 30),

stackView.leftAnchor.constraint(equalTo: mtkView.rightAnchor, constant: 30),

stackView.rightAnchor.constraint(equalTo: view.safeAreaLayoutGuide.rightAnchor, constant: -30),

stackView.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor, constant: -30),

])

}

//**MARK: - UIControl functions**

**@objc** **func** deltaValueChanged(\_ sender: UISlider!) {

**let** step = abs(approximationRange.0 - approximationRange.1) / sender.maximumValue

approximation = (sender.maximumValue + 1 - sender.value) \* step

}

**@objc** **func** stepperValueChanged(\_ sender: UIStepper!) {

rotation = rotations[Int(sender.value)]

rotationsLabel.text = "Rotations: \(rotation)"

}

**@objc** **func** pointValueChanged(\_ sender: UISlider!) {

points[sender.tag].y = Float(sender.value)

}

**@objc** **func** fillValueChanged(\_ sender: UISwitch!) {

**if** sender.isOn {

object.material.triangleFillMode = .fill

} **else** {

object.material.triangleFillMode = .lines

}

}

**@objc** **func** cullValueChanged(\_ sender: UISwitch!) {

**if** sender.isOn {

object.material.cullMode = .back

} **else** {

object.material.cullMode = MTLCullMode.none

}

}

//**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

renderer.rotationX -= xDelta

renderer.rotationY -= 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

renderer.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

}

}

RotationBezierCurve.swift

//

// RotationBezierCurve.swift

// CG\_KP

//

// Created by Илья Ильин on 25.12.2021.

//

**import** Foundation

**import** simd

**class** RotationBezierCurve : Node {

**var** controlPoints: [simd\_float3] {

**didSet** {

**self**.needRecount = **true**

}

}

**var** rotations: Int {

**didSet** {

**self**.needRecount = **true**

}

}

**init**(name: String, material: Material, controlPoints: [simd\_float3], approximation: Float, rotations: Int) {

**self**.controlPoints = controlPoints

**self**.rotations = rotations < 3 ? 3 : rotations

**super**.init(name: name, material: material, approximation: approximation)

}

**override** **func** countVertices() {

**let** bezierCurve = countBezierCurvePoints()

(**self**.vertices, **self**.indices) = countSurfaceVertices(bezierCurvePoints: bezierCurve)

print(">>> \(bezierCurve.first?.description ?? "none") \(bezierCurve.last?.description ?? "none")")

}

**private** **func** countSurfaceVertices(bezierCurvePoints: [simd\_float4]) -> ([Vertex], [[UInt16]]) {

**guard** bezierCurvePoints.count >= 3 **else** { **return** ([], []) }

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

**var** resultIndices = [UInt16]()

**var** shift: UInt16 = 0

**let** rotationMatrix = makeRotationMatrixAroundY(degrees: Double(360 / rotations))

**var** prevCurve = bezierCurvePoints

**for** rotation **in** 1...rotations {

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

**if** rotation == rotations {

curve = bezierCurvePoints

} **else** {

curve = prevCurve.map { rotationMatrix \* $0 }

}

///top triangle

**let** normal = countNormal(v1: prevCurve[0], v2: prevCurve[1], v3: curve[1])

**let** v0 = Vertex(point: prevCurve[0], normal: normal)

**let** v1 = Vertex(point: prevCurve[1], normal: normal)

**let** v2 = Vertex(point: curve[1], normal: normal)

resultVertices += [v0, v1, v2]

resultIndices += [shift, shift+1, shift+2]

shift += 3

/// others triangles

**var** prevLPoint = prevCurve[1]

**var** prevRPoint = curve[1]

**for** (lPoint, rPoint) **in** zip( prevCurve[2...], curve[2...] ) {

**let** normal1 = countNormal(v1: prevLPoint, v2: lPoint, v3: prevRPoint)

**let** normal2 = countNormal(v1: prevRPoint, v2: lPoint, v3: rPoint)

**let** v11 = Vertex(point: prevLPoint, normal: normal1)

**let** v12 = Vertex(point: lPoint, normal: normal1)

**let** v13 = Vertex(point: prevRPoint, normal: normal1)

**let** v21 = Vertex(point: prevRPoint, normal: normal2)

**let** v22 = Vertex(point: lPoint, normal: normal2)

**let** v23 = Vertex(point: rPoint, normal: normal2)

resultVertices += [v11, v12, v13, v21, v22, v23]

resultIndices += [shift+0, shift+1, shift+2, shift+3, shift+4, shift+5]

shift += 6

prevLPoint = lPoint

prevRPoint = rPoint

}

prevCurve = curve

}

**return** (resultVertices, [resultIndices])

}

**private** **func** countBinomial(n: Int, k: Int) -> Float {

**var** result: Float = 1.0

**if** (n == k || k < 1) {

**return** result

}

**for** i **in** 1...k {

result \*= Float( (n + 1 - i) / i )

}

**return** result

}

**private** **func** countBezierCurvePoints() -> [simd\_float4] {

**let** n = controlPoints.count - 1

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

**for** t **in** stride(from: Float(0.0), to: Float(1.0), by: approximation) {

**var** point = simd\_float3()

**for** i **in** 0...n {

point.x += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].x

point.y += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].y

point.z += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].z

}

bezierCurve.append( simd\_float4(point, 1.0) )

}

**let** t: Float = 1.0

**var** point = simd\_float3()

**for** i **in** 0...n {

point.x += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].x

point.y += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].y

point.z += countBinomial(n: n, k: i) \* pow(1 - t, Float(n - i)) \* pow(t, Float(i)) \* controlPoints[i].z

}

bezierCurve.append( simd\_float4(point, 1.0) )

**return** bezierCurve

}

}

1. Вывод

В ходе выполнения данного курсового проекта я закрепил свои знания из предыдущих ЛР, применил их для построения сложной параметрической поверхности и ее динамического изменения.

ЛИТЕРАТУРА

1. Документация Swift [Электронный ресурс] URL: https://swift.org/documentation/ (Дата обращения: 03.12.2021).
2. Bezier curve [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Bézier\_curve (дата обращения: 20.11.2021).