Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка песочницы Another Planet»**

**по ДПП ПП «Основы Gamedev и VR-разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Изучить работу библиотек | Бракк А. И.  Семенков Н. Е.  Берлова К. А. | 02.05.2025 – 13.05.2025 |
| 2 | Разработать видеоигру | 14.05.2025 – 18.05.2025 |
| 3 | Провести тестирование | 19.05.2025 – 26.05.2025 |
| 4 | Подготовка отчета и видео-презентации | 26.05.2025 – 30.05.2025 |

Руководитель проекта   
преподаватель Осыкин Д. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д. Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«**Основы Gamedev и VR-разработки»**

**«Разработка песочницы Another Planet»**

Исполнители:

Бракк А. И.

Семенков Н. Е.

Берлова К. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

Осыкин Д. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Цель данного проекта — разработка видеоигры AnotherPlanet, представляющей собой инновационную воксельную песочницу, которая подарит игрокам уникальный, живой и динамичный трёхмерный мир, наполненный бескрайними возможностями для исследования, творчества и самовыражения. AnotherPlanet стремится стать платформой, где каждый пользователь сможет почувствовать полную свободу действий, погрузиться в атмосферу глубокого взаимодействия с окружением и реализовать свои самые смелые идеи в строительстве и выживании.

# Задачи проекта и исполнители

Для успешного достижения поставленной цели по созданию видеоигры AnotherPlanet необходимо решить комплекс взаимосвязанных задач, охватывающих все этапы разработки — от предварительного исследования и проектирования до выпуска и поддержки продукта.

Основные задачи проекта включают:

* Изучение и анализ технологий.
* Проектирование архитектуры игры.
* Разработка видеоигры.
* Тестирование и отладка.

Исполнители проекта:

* Бракк Артём.
* Берлова Ксения.
* Семенков Никита.

# Актуальность и востребованность проекта (разрабатываемого продукта)

Воксельные песочницы остаются одними из самых востребованных игровых жанров благодаря открытому миру, свободе творчества и исследованию.

Ее востребованность подтверждается следующими ключевыми факторами:

1. Устойчивая популярность жанра

* Рост рынка воксельных игр: Годовой рост жанра составляет 12-15% (Newzoo, 2023), 78% игроков в возрасте 12-35 лет регулярно играют в песочницы (Statista, 2024), Среднее время сессии: 2,7 часа (Steam Analytics).
* Воксельные песочницы остаются одним из самых стабильно популярных жанров на протяжении последнего десятилетия.
* Примеры успешных проектов: Minecraft - 238 млн проданных копий (2023), Terraria - 44 млн копий, Space Engineers - 5 млн копий.
* Средний срок жизни подобных проектов превышает 5-7 лет благодаря постоянным обновлениям и модификациям.

1. Популярность космической тематики

* Рост интереса к космической тематике в поп-культуре: Успех No Man's Sky (20 млн игроков), Популярность Starfield (10 млн игроков за 3 недели после релиза), Востребованность космических симуляторов (Elite Dangerous, Star Citizen).

1. Технологическая актуальность

* Использование современных технологий

1. Коммерческие перспективы

* Объем рынка воксельных игр оценивается в $3.2 млрд (2023).
* Средний ROI успешных проектов жанра составляет 300-500%.
* Возможности монетизации:
  + Продажа полной версии.
  + Косметические DLC.
  + Система модификаций.
  + Лицензии.

# Общие сведения о проделанной работе

Для разработки был выбран язык программирования Python, так как он содержит все необходимые для проекта библиотеки, он легок в освоении.

Для визуализации работы будет использоваться библиотека Pygame, потому что на ней можно отлично реализовать и визуально воспроизвести игровой процесс.

Также будет использоваться библиотеки:

Math для проведения различных тригонометрических вычислений.

ModernGL. Основная роль — эффективный рендеринг расчётов, выполненных в других библиотеках.

PyGLM — это специализированная библиотека для 3D-математики на Python, которая предоставляет полный набор функций и типов данных, необходимых для работы с трехмерной графикой и игровым программированием. Библиотека включает glm.sin(), glm.cos(), повороты векторов и матричные преобразования. Она идеальна для камеры, коллизий и анимаций.

NumPy является классическим выбором для массовых вычислений. NumPy — это фундаментальная библиотека для научных и численных вычислений в Python. Она предоставляет мощные инструменты для работы с многомерными массивами и матрицами, а также широкий набор математических функций.

Разработка игры началась с идеи создать игру в жанре воксельная песочница. был разработан работающий прототип, который реализован с использованием библиотек языка Python, которые были описаны выше.

Бала разработана генерация карта. Мир в игре состоит из чанков – отдельных блоков карты, которые загружаются и выгружаются динамически. Это позволяет эффективно управлять памятью и снижать нагрузку на систему.

Для оптимизации рендеринга реализованы следующие механизмы:

Отсечение невидимых блоков – рендерятся только те блоки, которые находятся в поле зрения камеры.

Окклюзия – если блок полностью закрыт другими, он не отправляется на отрисовку.

Асинхронная загрузка чанков – генерация новых участков карты происходит в фоновом режиме, предотвращая подтормаживания.

Благодаря этим методам игра демонстрирует стабильный FPS даже на средних по мощности компьютерах.

В игре реализована возможность разрушения блоков и строительство из блоков. Игрок может удалять воксели и размещать новые, изменяя окружающий мир.

Реализована перемещение персонажа, с помощью клавиатура, и вращение камерой, с помощью мыши.

Чтобы добавить атмосферности, были в игру добавлены облака, которые строится с помощью простейших фигур. Реализовано перемещение облаков.

Чтобы добавить немного детализации, были созданы тени от блоков.

После игра была протестирована. Она прошла базовое тестирование на

различных конфигурациях ПК.

# Результаты проекта

При запуске видеоигры отображается карта, с генерируемая с помощью математический вычислений, состоящая из разных уровней ландшафта.

Сгенерированный мир изображён на рисунке 1.

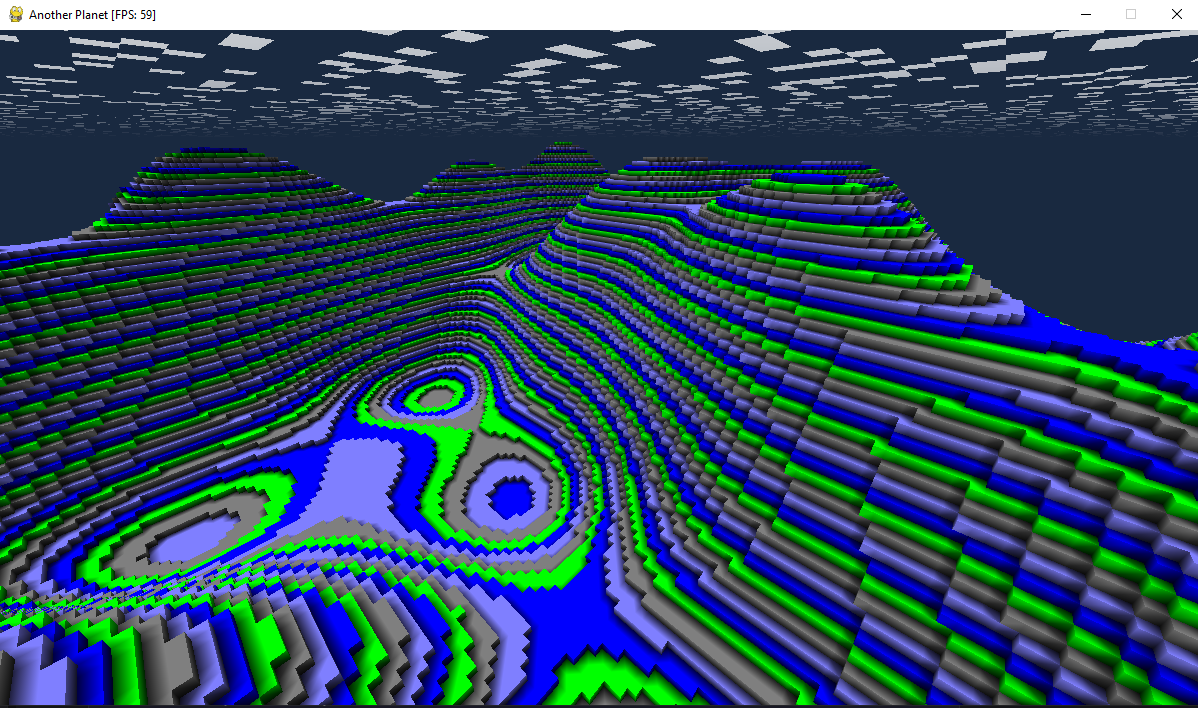


Рисунок 1 – Сгенерированный мир

В видеоигре есть возможность строительство различных зданий или других построек. Постройки изображены на рисунке 2.

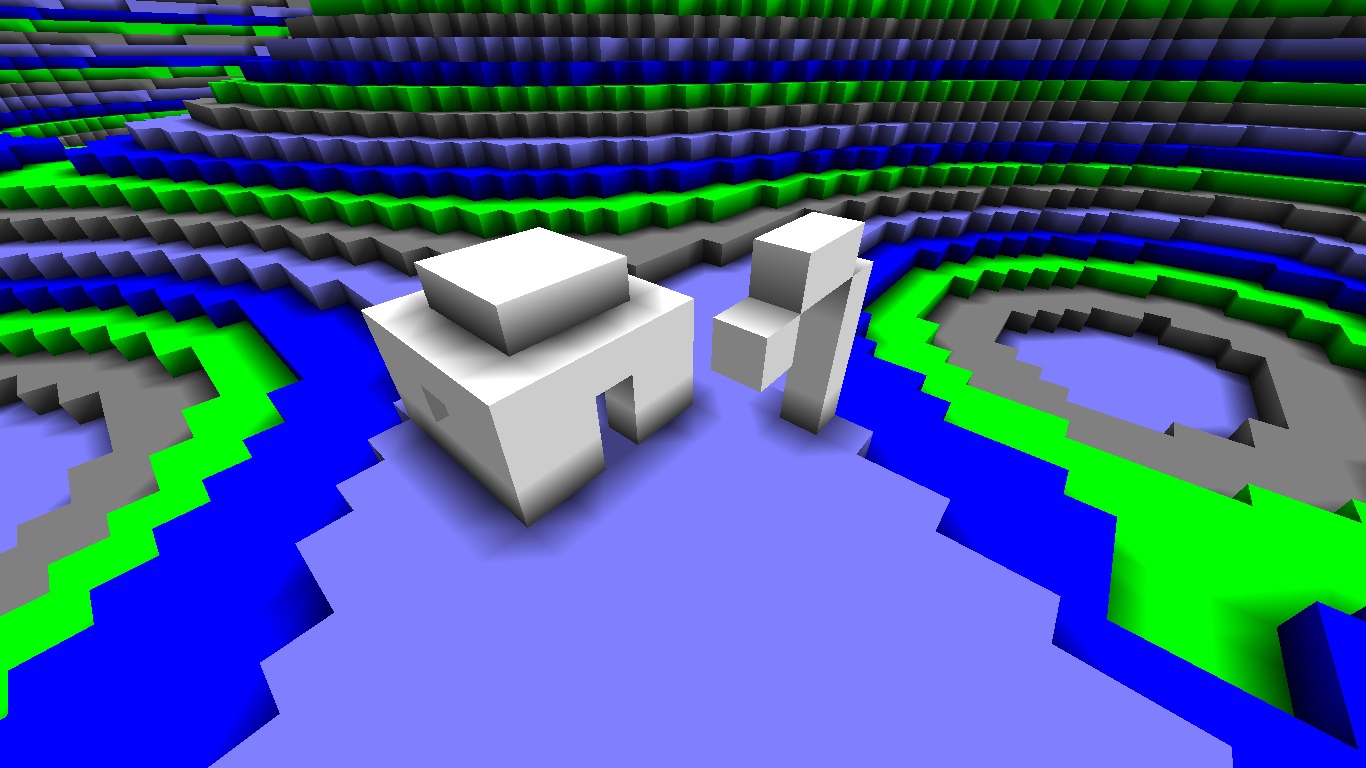


Рисунок 2 – Постройки

В игре создано отображение теней – когда от блока падает тень. Тени от блоков изображены на рисунке 3.

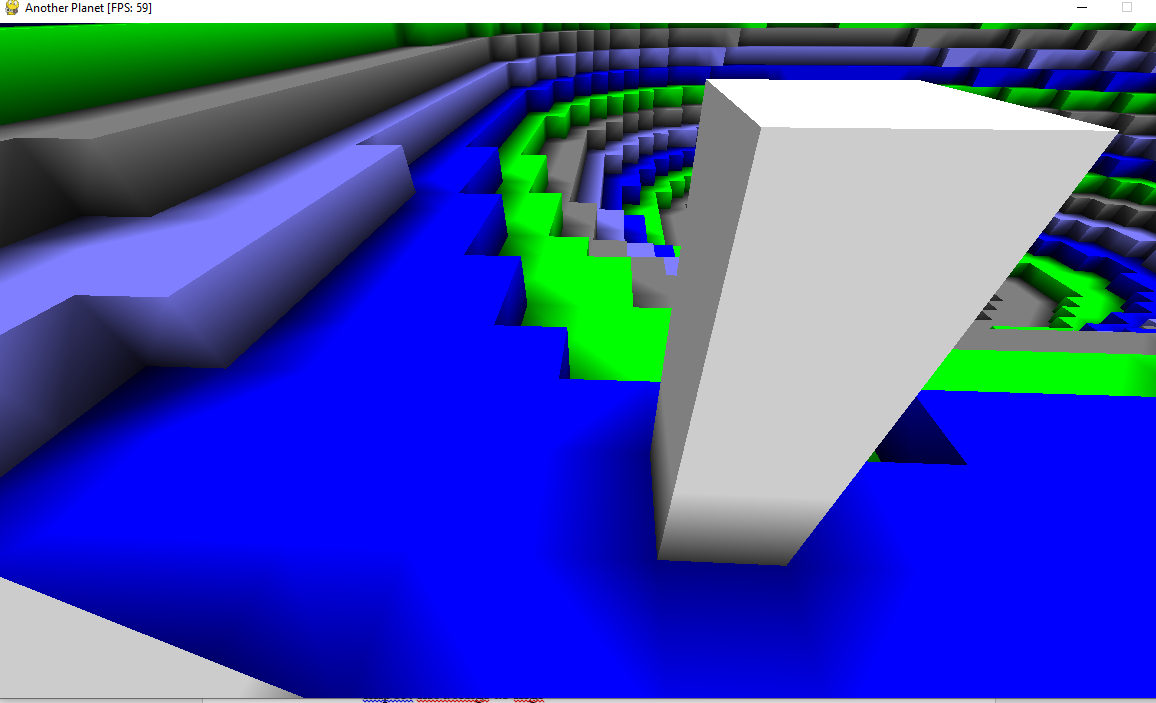


Рисунок 3 – Тень

# Приложение 1.

Программный код

from settings import gl\_settings, common\_settings

import moderngl as mgl

import pygame as pg

import sys

from scene import Scene

from shader import Shader

from player import Player

class Engine:

def \_\_init\_\_(self):

pg.init()

pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_MAJOR\_VERSION, gl\_settings.GL\_MAJOR\_VERSION)

pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_MAJOR\_VERSION, gl\_settings.GL\_MINOR\_VERSION)

pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_PROFILE\_MASK, pg.GL\_CONTEXT\_PROFILE\_CORE)

pg.display.gl\_set\_attribute(pg.GL\_DEPTH\_SIZE, gl\_settings.GL\_DEPTH\_BUFFER\_SIZE)

pg.display.set\_mode(common\_settings.WIN\_RES, flags=pg.OPENGL | pg.DOUBLEBUF)

#pg.display.set\_mode((0, 0), flags=pg.OPENGL | pg.DOUBLEBUF | pg.FULLSCREEN)

self.ctx = mgl.create\_context()

self.ctx.enable(flags=mgl.DEPTH\_TEST | mgl.CULL\_FACE | mgl.BLEND)

self.ctx.gc\_mode = 'auto'

self.clock = pg.time.Clock()

self.time = 0

self.dt = 0

pg.event.set\_grab(True)

pg.mouse.set\_visible(False)

self.player = None

self.shader = None

self.scene = None

self.running = False

def on\_init(self):

# Player should be init before all others!

self.player = Player(self)

# Shader should be init before World!

self.shader = Shader(self)

self.scene = Scene(self)

def update(self):

self.player.update()

self.shader.update()

self.scene.update()

self.dt = self.clock.tick()

self.time = pg.time.get\_ticks() \* 0.001

def render(self):

self.ctx.clear(color=common\_settings.BACKGROUND\_COLOR)

self.scene.render()

pg.display.flip()

def event(self):

for event in pg.event.get():

if event.type == pg.QUIT or (event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE):

self.running = False

self.player.event(event)

def start(self):

self.running = True

self.on\_init()

prev\_second\_ticks = pg.time.get\_ticks()

unprocessed\_ticks = 0

now\_ticks = pg.time.get\_ticks()

prev\_ticks = now\_ticks

target\_update\_delta = 1000 / 60

while self.running:

self.event()

now\_ticks = pg.time.get\_ticks()

unprocessed\_ticks += (now\_ticks - prev\_ticks)

prev\_ticks = now\_ticks

if unprocessed\_ticks >= target\_update\_delta:

self.update()

self.render()

unprocessed\_ticks = 0

if pg.time.get\_ticks() - prev\_second\_ticks > 1000:

pg.display.set\_caption(f'Another Planet [FPS: {self.clock.get\_fps() :.0f}]')

prev\_second\_ticks = pg.time.get\_ticks()

pg.quit()

sys.exit()

if \_\_name\_\_ == '\_\_main\_\_':

engine = Engine()

engine.start()

import glm

from frustum import Frustum

from settings import common\_settings

class Camera:

def \_\_init\_\_(self, position, yaw, pitch):

self.position = glm.vec3(position)

self.yaw = glm.radians(yaw)

self.pitch = glm.radians(pitch)

self.up = glm.vec3(0, 1, 0)

self.right = glm.vec3(1, 0, 0)

self.forward = glm.vec3(0, 0, -1)

self.m\_proj = glm.perspective(common\_settings.V\_FOV,

common\_settings.ASPECT\_RATIO,

common\_settings.NEAR,

common\_settings.FAR)

self.m\_view = glm.mat4()

self.frustum = Frustum(self)

def update(self):

self.update\_vectors()

self.update\_view\_matrix()

def update\_view\_matrix(self):

self.m\_view = glm.lookAt(self.position, self.position + self.forward, self.up)

def update\_vectors(self):

self.forward.x = glm.cos(self.yaw) \* glm.cos(self.pitch)

self.forward.y = glm.sin(self.pitch)

self.forward.z = glm.sin(self.yaw) \* glm.cos(self.pitch)

self.forward = glm.normalize(self.forward)

self.right = glm.normalize(glm.cross(self.forward, glm.vec3(0, 1, 0)))

self.up = glm.normalize(glm.cross(self.right, self.forward))

def rotate\_pitch(self, delta\_y):

self.pitch -= delta\_y

self.pitch = glm.clamp(self.pitch, -common\_settings.PITCH\_MAX, common\_settings.PITCH\_MAX)

def rotate\_yaw(self, delta\_x):

self.yaw += delta\_x

def move\_left(self, velocity):

self.position -= self.right \* velocity

def move\_right(self, velocity):

self.position += self.right \* velocity

def move\_up(self, velocity):

self.position += self.up \* velocity

def move\_down(self, velocity):

self.position -= self.up \* velocity

def move\_forward(self, velocity):

self.position += self.forward \* velocity

def move\_back(self, velocity):

self.position -= self.forward \* velocity

from settings.common\_settings import \*

class Frustum:

def \_\_init\_\_(self, camera):

from camera import Camera

self.camera: Camera = camera

self.factor\_y = 1.0 / math.cos(half\_y := V\_FOV \* 0.5)

self.tan\_y = math.tan(half\_y)

self.factor\_x = 1.0 / math.cos(half\_x := H\_FOV \* 0.5)

self.tan\_x = math.tan(half\_x)

def is\_on\_frustum(self, chunk):

# vector to sphere center

sphere\_vector = chunk.center\_position - self.camera.position

# outside the NEAR and FAR planes?

sz = glm.dot(sphere\_vector, self.camera.forward)

if not (NEAR - CHUNK\_SPHERE\_RADIUS <= sz <= FAR + CHUNK\_SPHERE\_RADIUS):

return False

# outside the TOP and BOTTOM planes?

sy = glm.dot(sphere\_vector, self.camera.up)

dist = self.factor\_y \* CHUNK\_SPHERE\_RADIUS + sz \* self.tan\_y

if not (-dist <= sy <= dist):

return False

# outside the LEFT and RIGHT planes?

sx = glm.dot(sphere\_vector, self.camera.right)

dist = self.factor\_x \* CHUNK\_SPHERE\_RADIUS + sz \* self.tan\_x

if not (-dist <= sx <= dist):

return False

return True

import pygame as pg

from camera import Camera

from settings.common\_settings import \*

class Player(Camera):

def \_\_init\_\_(self, engine, position=PLAYER\_POS, yaw=-90, pitch=0):

self.engine = engine

super().\_\_init\_\_(position, yaw, pitch)

def keyboard\_control(self):

key\_state = pg.key.get\_pressed()

velocity = PLAYER\_SPEED \* self.engine.dt

if key\_state[pg.K\_w]:

self.move\_forward(velocity)

if key\_state[pg.K\_d]:

self.move\_right(velocity)

if key\_state[pg.K\_s]:

self.move\_back(velocity)

if key\_state[pg.K\_a]:

self.move\_left(velocity)

if key\_state[pg.K\_e]:

# self.move\_down(velocity)

self.position -= glm.vec3(0.0, -1.0, 0.0) \* velocity

if key\_state[pg.K\_q]:

# self.move\_up(velocity)

self.position -= glm.vec3(0.0, 1.0, 0.0) \* velocity

if key\_state[pg.K\_LEFT]:

self.rotate\_yaw(-0.05)

if key\_state[pg.K\_RIGHT]:

self.rotate\_yaw(0.05)

if key\_state[pg.K\_UP]:

self.rotate\_pitch(-0.05)

if key\_state[pg.K\_DOWN]:

self.rotate\_pitch(0.05)

def update(self):

self.keyboard\_control()

self.mouse\_control()

super().update()

def event(self, event):

if event.type == pg.MOUSEBUTTONDOWN:

voxel\_handler = self.engine.scene.world.voxel\_handler

if event.button == pg.BUTTON\_LEFT:

voxel\_handler.remove\_voxel()

if event.button == pg.BUTTON\_RIGHT:

voxel\_handler.add\_voxel()

def mouse\_control(self):

mouse\_dx, mouse\_dy = pg.mouse.get\_rel()

if mouse\_dx:

self.rotate\_yaw(delta\_x=mouse\_dx \* MOUSE\_SENSITIVITY)

if mouse\_dy:

self.rotate\_pitch(delta\_y=mouse\_dy \* MOUSE\_SENSITIVITY)

import moderngl

from settings.common\_settings import \*

from world.world import World

from world.objects.clouds import Clouds

class Scene:

def \_\_init\_\_(self, engine):

self.engine = engine

self.world = World(self.engine)

self.clouds = Clouds(self.engine)

def update(self):

self.world.update()

self.clouds.update()

def render(self):

self.world.render()

self.engine.ctx.disable(moderngl.CULL\_FACE)

self.clouds.render()

self.engine.ctx.enable(moderngl.CULL\_FACE)

from settings.common\_settings import \*

class Shader:

def \_\_init\_\_(self, engine):

self.engine = engine

self.ctx = engine.ctx

self.player = engine.player

self.chunk = self.get\_program('chunk')

self.clouds = self.get\_program('clouds')

self.set\_uniform\_on\_init()

def set\_uniform\_on\_init(self):

self.chunk['m\_proj'].write(self.player.m\_proj)

self.chunk['m\_model'].write(glm.mat4())

self.clouds['m\_proj'].write(self.player.m\_proj)

self.clouds['center'] = WORLD\_CENTER\_XZ

self.clouds['bg\_color'].write(BACKGROUND\_COLOR)

self.clouds['cloud\_scale'] = 24

def update(self):

self.chunk['m\_view'].write(self.player.m\_view)

self.clouds['m\_view'].write(self.player.m\_view)

def get\_program(self, shader\_name):

with open(f"resources/shaders/{shader\_name}\_vert.glsl") as file:

vertex\_shader = file.read()

with open(f"resources/shaders/{shader\_name}\_frag.glsl") as file:

fragment\_shader = file.read()

program = self.ctx.program(vertex\_shader=vertex\_shader, fragment\_shader=fragment\_shader)

return program

from settings.common\_settings import \*

from mesh.chunk\_mesh\_builder import get\_chunk\_index

class VoxelHandler:

def \_\_init\_\_(self, world):

self.engine = world.engine

self.chunks = world.chunks

self.chunk = None

self.voxel\_id = None

self.voxel\_index = None

self.voxel\_local\_pos = None

self.voxel\_world\_pos = None

self.voxel\_normal = None

self.new\_voxel\_id = 5

def add\_voxel(self):

if self.voxel\_id:

result = self.get\_voxel\_id(self.voxel\_world\_pos + self.voxel\_normal)

# TODO FUCKING SHIT

# print(self.voxel\_world\_pos)

if self.voxel\_world\_pos[1] >= WORLD\_H \* CHUNK\_SIZE:

return

if not result[0]:

\_, voxel\_index, \_, chunk = result

chunk.voxels[voxel\_index] = self.new\_voxel\_id

chunk.mesh.rebuild()

if chunk.is\_empty:

chunk.is\_empty = False

def rebuild\_adj\_chunk(self, adj\_voxel\_pos):

index = get\_chunk\_index(adj\_voxel\_pos)

if index != -1:

self.chunks[index].mesh.rebuild()

def rebuild\_adjacent\_chunks(self):

lx, ly, lz = self.voxel\_local\_pos

wx, wy, wz = self.voxel\_world\_pos

if lx == 0:

self.rebuild\_adj\_chunk((wx - 1, wy, wz))

elif lx == CHUNK\_SIZE - 1:

self.rebuild\_adj\_chunk((wx + 1, wy, wz))

if ly == 0:

self.rebuild\_adj\_chunk((wx, wy - 1, wz))

elif ly == CHUNK\_SIZE - 1:

self.rebuild\_adj\_chunk((wx, wy + 1, wz))

if lz == 0:

self.rebuild\_adj\_chunk((wx, wy, wz - 1))

elif lz == CHUNK\_SIZE - 1:

self.rebuild\_adj\_chunk((wx, wy, wz + 1))

def remove\_voxel(self):

if self.voxel\_id:

self.chunk.voxels[self.voxel\_index] = 0

self.chunk.mesh.rebuild()

self.rebuild\_adjacent\_chunks()

def update(self):

self.ray\_cast()

def ray\_cast(self):

x1, y1, z1 = self.engine.player.position

x2, y2, z2 = self.engine.player.position + self.engine.player.forward \* RAYCASTING\_MAX\_LENGTH

current\_voxel\_pos = glm.ivec3(x1, y1, z1)

self.voxel\_id = 0

self.voxel\_normal = glm.ivec3(0)

step\_dir = 1

dx = glm.sign(x2 - x1)

delta\_x = min(dx / (x2 - x1), 10000000.0) if dx != 0 else 10000000.0

max\_x = delta\_x \* (1.0 - glm.fract(x1)) if dx > 0 else delta\_x \* glm.fract(x1)

dy = glm.sign(y2 - y1)

delta\_y = min(dy / (y2 - y1), 10000000.0) if dy != 0 else 10000000.0

max\_y = delta\_y \* (1.0 - glm.fract(y1)) if dy > 0 else delta\_y \* glm.fract(y1)

dz = glm.sign(z2 - z1)

delta\_z = min(dz / (z2 - z1), 10000000.0) if dz != 0 else 10000000.0

max\_z = delta\_z \* (1.0 - glm.fract(z1)) if dz > 0 else delta\_z \* glm.fract(z1)

while not (max\_x > 1.0 and max\_y > 1.0 and max\_z > 1.0):

result = self.get\_voxel\_id(voxel\_world\_pos=current\_voxel\_pos)

if result[0]:

self.voxel\_id, self.voxel\_index, self.voxel\_local\_pos, self.chunk = result

self.voxel\_world\_pos = current\_voxel\_pos

if step\_dir == 0:

self.voxel\_normal.x = -dx

elif step\_dir == 1:

self.voxel\_normal.y = -dy

else:

self.voxel\_normal.z = -dz

return True

if max\_x < max\_y:

if max\_x < max\_z:

current\_voxel\_pos.x += dx

max\_x += delta\_x

step\_dir = 0

else:

current\_voxel\_pos.z += dz

max\_z += delta\_z

step\_dir = 2

else:

if max\_y < max\_z:

current\_voxel\_pos.y += dy

max\_y += delta\_y

step\_dir = 1

else:

current\_voxel\_pos.z += dz

max\_z += delta\_z

step\_dir = 2

return False

def get\_voxel\_id(self, voxel\_world\_pos):

cx, cy, cz = chunk\_pos = voxel\_world\_pos / CHUNK\_SIZE

if 0 <= cx < WORLD\_W and 0 <= cy < WORLD\_H and 0 <= cz < WORLD\_D:

chunk\_index = cx + WORLD\_W \* cz + WORLD\_AREA \* cy

chunk = self.chunks[chunk\_index]

lx, ly, lz = voxel\_local\_pos = voxel\_world\_pos - chunk\_pos \* CHUNK\_SIZE

voxel\_index = lx + CHUNK\_SIZE \* lz + CHUNK\_AREA \* ly

voxel\_id = chunk.voxels[voxel\_index]

return voxel\_id, voxel\_index, voxel\_local\_pos, chunk

return 0, 0, 0, 0

# Приложение 2.

Ссылка на видео: <https://drive.google.com/drive/folders/1ZYMFswtyv2El6FTtpeaYwzqF24Bea5eX?usp=sharing>

# Приложение 3.

Ссылка на презентацию (защита):

<https://drive.google.com/drive/folders/1j_oRbE0_AAcDZV-B0cOlrX1lct-yvX2V?usp=sharing>

# Приложение 4.

Ссылка на файл программы: <https://drive.google.com/drive/folders/1b4gBVCFb-xLPzHC9clMZ_AL4_GN2i16w?usp=sharing>