ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		И.М. Лозоватский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Создание рельефа, рендеринг больших изображений

по дисциплине: Проектирование человеко-машинного интерфейса

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4134к		Костяков Н.А.
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

Цель работы: освоение способов создания рельефа в Blender, изучение возможностей скриптового языка Python, настройка параметров рендеринга.

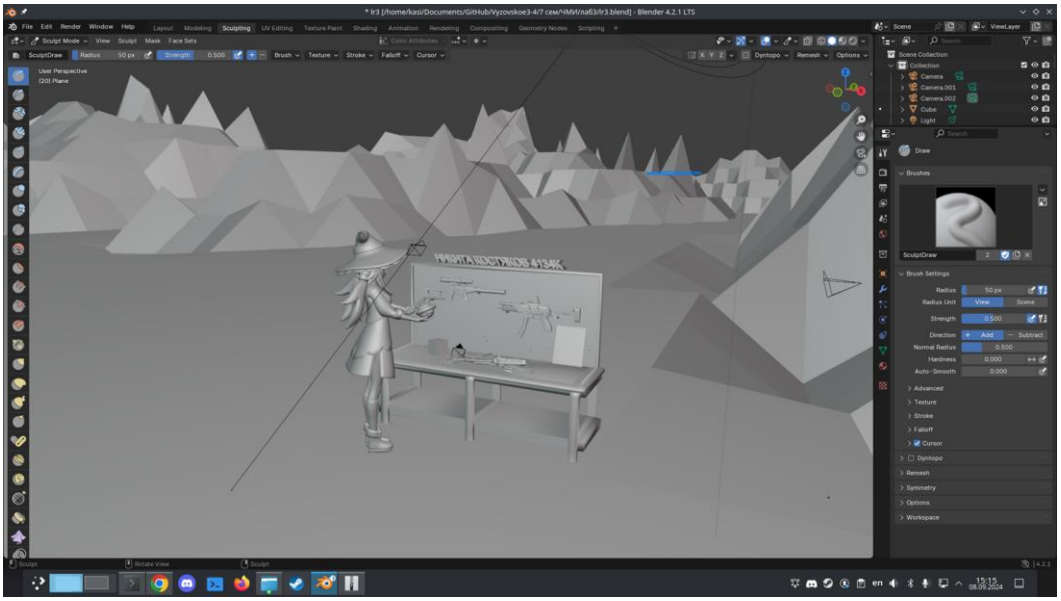
Задание: Создать рельеф двумя различными способами:
1. Сгенерировать рельеф на основе карты высот с помощью модификаторов: а) создать объект Plane; б) применить к нему один или несколько модификаторов SubSurf, чтобы увеличить число вершин с 4 до нескольких сотен или тысяч; в) после SubSurf применить модификатор Displace, указав в поле Texture текстуру, являющуюся картой высот

2. С помощью скрипта Blender World Forge Tool (BWF) сгенерировать рельеф. Добавить кратеры и пики. Для того чтобы воспользоваться скриптом необходимо следующее: а) Установить Python соответствующей Blender'у версии. б) Перезапустить Blender и убедиться, что в консоли появилась строка Checking for installed Python... got it! в) Открыть в Blender’е окно типа Text Editor г) Alt+O – открыть текстовый файл (выбрать файл скрипта BWF-0.1.0.py) д) Alt+P – выполнить скрипт е) в окне, где был Text Editor, появится новое окошко, в котором необходимо задать настройки генерируемого рельефа и нажать кнопку TERRAFORM Настроить параметры рендеринга, получить изображение не хуже 1600*1200 pix, осуществить рендеринг в файл в формате JPEG.

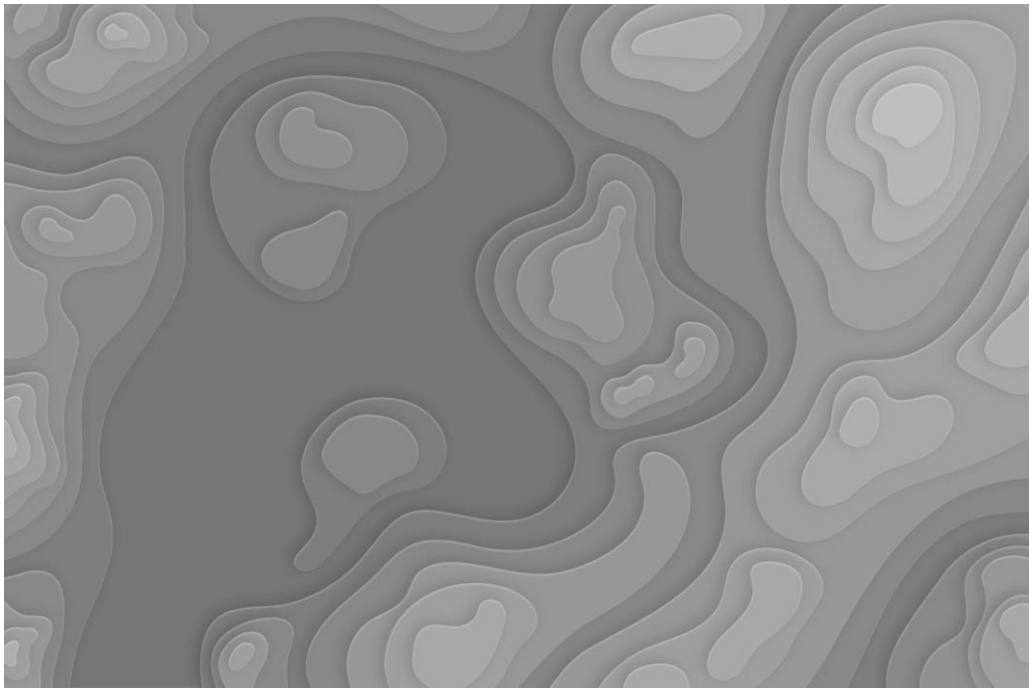
Словесное описание сцены

Верстрак, на котором лежат оружейные компоненты, ВСС-Винторез, ПП-19-01, рядом с верстаком модель человека, который держит в руках револьвер и композицию из первой лабораторной работы

Описание технологии создания сцены
Загрузить модели с любого инттернет ресурса в формате fbl и импортировать их в проект. Отредактировать положение и размеры при помощи встроенных инструментов. Надпись сделана посредством копирования и вставки кода символа в объект text. Перейти в панель Shading, при импорте моделей в формате fbl их текстуры и материалы сразу будут отображены шейдерами. Базовый куб хоть и выглядит бестекстурным, но уже отображается коректно. В добавленные стандартные фигуры нужно указать параметры материалов. Вводом разых чисел в параметры Roughnes, Metallic, IOR, Alpha можно добиться разных свойст для отраженного света. Создав объект plane применил модификаторы, чтобы получить рельеф по текстуре ниже



Результат генерации ландшафта по текстуре



Текстура

Скрипт для генерации ландшафта на python

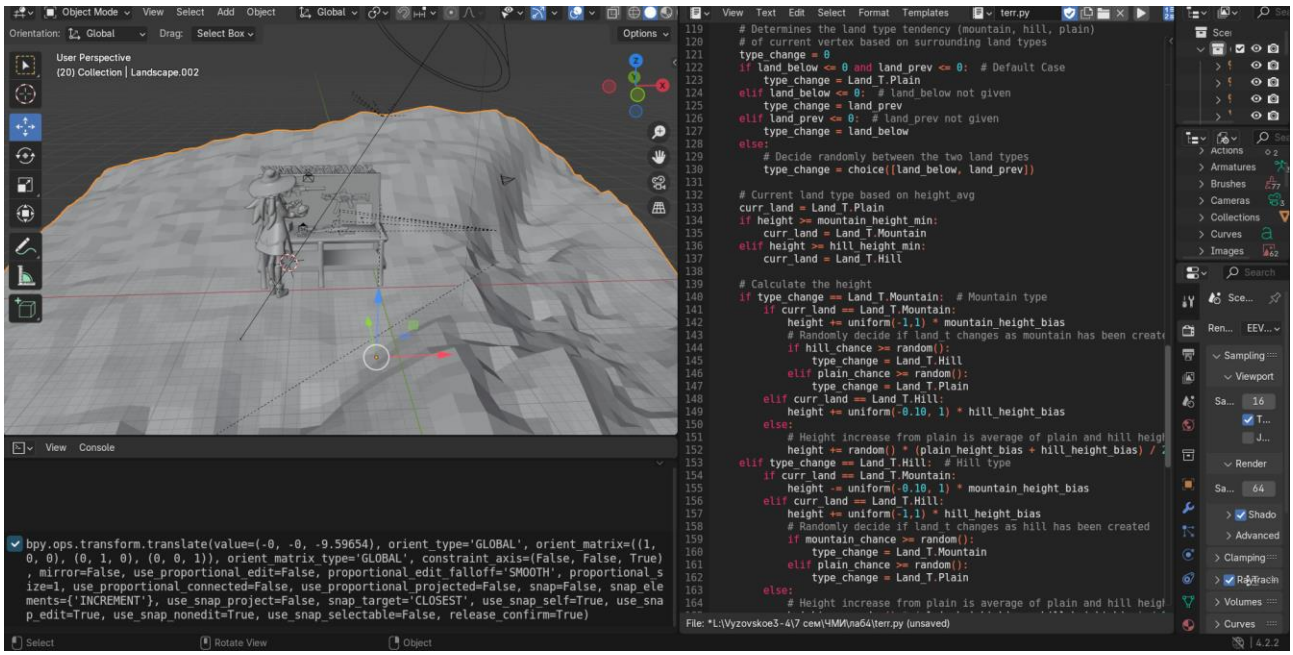
```
import bpy
from random import *

# Number of terrain subdivisions (should be > 50)
subdiv = 50
# Total length and width of terrain
size = 100
# Maximum height of any vertex
height_max = 50
```

```
# Minimum height to be considered a mountain/hill
mountain_height_min = height_max * (6/10.0)
hill_height_min = height_max * (2/10.0)

# Maximum height for a plain (should be less than hill height)
plain_height_max = height_max * (1/80.0)

# Chance of mountains/hills biases being added/subtracted
mountain_chance = 0.025
hill_chance =  0.050
plain_chance = 0.750

# Maximum jump in height from one vertex to the next
# (For good results: plain < hill < mountain, and smaller for higher subdiv)
mountain_height_bias = height_max * (1/40.0)
hill_height_bias = height_max * (1/60.0)
plain_height_bias = plain_height_max * (1/1.0)


def main():
    verts, faces = create_landscape()
    edges = []

    # Create new mesh
    mesh = bpy.data.meshes.new("Landscape_Data")

    # Make a mesh from a list of verts/edges/faces.
    mesh.from_pydata(verts, edges, faces)

    # Update mesh geometry after adding stuff.
    mesh.update()

    object = bpy.data.objects.new("Landscape", mesh)
    object.data = mesh

    # Link mesh to scene
    scene = bpy.context.scene
    scene.objects.link(object)
    object.select = True


# Calculate the vertices and faces of a terrain and return them
def create_landscape():
    verts = []
    faces = []
    # List of each vertex's tendency towards a land type (Land_T)
    land_tend = []

    start_loc = -(size / 2.0)
    face_size = size/ float(subdiv - 1)
```

```
    # Walk through the rows and columns of a square plane
    # and create their vertices
    for row in range(subdiv):  # Row in x-direction
        x = start_loc + row * face_size
        for col in range(subdiv):  # Column in y-direction
            y = start_loc + col * face_size
            # Need to calculate height and Land_T of vertices by
            # observing height of surrounding vertices
            height_average = 0
            curr_land_t = Land_T.Plain
            # Use height of previous row
            if row > 0:
                vert_below = verts[col + subdiv * (row - 1)]
                land_below = land_tend[col + subdiv * (row - 1)]
                # Use height of previous vertex in current edge loop
                if col > 0:
                    vert_prev = verts[(col - 1) + subdiv * row]
                    land_prev = land_tend[(col - 1) + subdiv * row]
                    height_average = (vert_prev[2] + vert_below[2]) / 2.0
                    z, curr_land_t = height_gen(height_average, land_below,
land_prev)

                    # Create a face, using indices of vertices
                    vertInd1 = (col - 1) + subdiv * row
                    vertInd2 = col + subdiv * row
                    vertInd3 = (col - 1) + subdiv * (row - 1)
                    vertInd4 = col + subdiv * (row - 1)
                    face_new = (vertInd2, vertInd1, vertInd3, vertInd4)
                    faces.append(face_new)
                # First vertex of each row
                else:
                    height_average = vert_below[2]
                    z, curr_land_t = height_gen(height_average, land_below, 0)
            # First edge loop
            else:
                if col > 0:
                    vert_prev = verts[col - 1]
                    land_prev = land_tend[col - 1]
                    height_average = vert_prev[2]
                    z, curr_land_t = height_gen(height_average, 0, land_prev)
                # First vertex
                else:
                    # Random starting height average of plain type
                    height_average = uniform(0, plain_height_max)
                    z, curr_land_t = height_gen(height_average, Land_T.Plain, 0)

            verts.append((x,y,z))
            land_tend.append(curr_land_t)

    return verts, faces
```

```
# Calculates a random height based on the average height and the
# land type tendencies of the vertices surrounding it
def height_gen(height_avg, land_below, land_prev):
    height = height_avg

    # Determines the land type tendency (mountain, hill, plain)
    # of current vertex based on surrounding land types
    type_change = 0
    if land_below <= 0 and land_prev <= 0:  # Default Case
        type_change = Land_T.Plain
    elif land_below <= 0:  # land_below not given
        type_change = land_prev
    elif land_prev <= 0:  # land_prev not given
        type_change = land_below
    else:
        # Decide randomly between the two land types
        type_change = choice([land_below, land_prev])

    # Current land type based on height_avg
    curr_land = Land_T.Plain
    if height >= mountain_height_min:
        curr_land = Land_T.Mountain
    elif height >= hill_height_min:
        curr_land = Land_T.Hill

    # Calculate the height
    if type_change == Land_T.Mountain:  # Mountain type
        if curr_land == Land_T.Mountain:
            height += uniform(-1,1) * mountain_height_bias
            # Randomly decide if land_t changes as mountain has been created
            if hill_chance >= random():
                type_change = Land_T.Hill
            elif plain_chance >= random():
                type_change = Land_T.Plain
        elif curr_land == Land_T.Hill:
            height += uniform(-0.10, 1) * hill_height_bias
        else:
            # Height increase from plain is average of plain and hill height bias
            height += random() * (plain_height_bias + hill_height_bias) / 2.0
    elif type_change == Land_T.Hill:  # Hill type
        if curr_land == Land_T.Mountain:
            height -= uniform(-0.10, 1) * mountain_height_bias
        elif curr_land == Land_T.Hill:
            height += uniform(-1,1) * hill_height_bias
            # Randomly decide if land_t changes as hill has been created
            if mountain_chance >= random():
                type_change = Land_T.Mountain
            elif plain_chance >= random():
                type_change = Land_T.Plain
        else:
            # Height increase from plain is average of plain and hill height bias
```

```
            height += random() * (plain_height_bias + hill_height_bias) / 2.0
    else:  # Plain type
        if curr_land == Land_T.Mountain:
            height -= uniform(-0.10, 1) * mountain_height_bias
        elif curr_land == Land_T.Hill:
            height += uniform(-1,1) * hill_height_bias
        else:
            height += uniform(-1,1) * plain_height_bias
            # Clamp plain height to max
            if height >= plain_height_max:
                height -= 2* random() * plain_height_bias
            # Randomly decide if land_t changes as plain has been created
            if mountain_chance >= random():
                type_change = Land_T.Mountain
            elif hill_chance >= random():
                type_change = Land_T.Hill

    # Clamp height to max and minimum height
    if height > height_max:
        height = height_max
    elif height < 0:
        height = 0
    return height, type_change


# A faux-enumerator that represents land type
class Land_T:
    Mountain = 1
    Hill = 2
    Plain = 3

if __name__ == "__main__":
    main()
```



Пример результата работы скрипта генерации

Пример с рендером ландшафта

Выводы

Я освоил способы построения рельефа в blender