УДК 004.041, 004.925.84

ПРОЦЕДУРНАЯ ГЕНЕРАЦИЯ ТРЕХМЕРНЫХ ПРИМИТИВНЫХ ОБЪЕКТОВ

Селезнев А.С.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина, г. Рязань, Российская Федерация

Научный руководитель: Бакулев А.В - к.т.н., доцент, доцент кафедры САПР ВС
Бакулева М.А. - к.т.н., доцент, доцент кафедры САПР ВС

Аннотация. В статье рассмотрены применение различных методов процедурной генерации трехмерных объектов, использующихся в настоящее время, а также реализацию некоторых алгоритмов на языке Python.

Ключевые слова: процедурная генерация, 3D моделирование, Python, программирование

Введение. 3D-моделирование в современном мире активно применяется для решения разных задач. Это процесс, требующий творческого подхода, большого объема навыков и знаний, также предполагает долгий и упорный труд. Часто требуется создавать множество схожих, но уникальных трехмерных моделей. Они могут быть востребованы в различных областях, начиная с архитектурной (генерация зданий, местностей и так далее), заканчивая игровой индустрией, где их используют для обогащения игрового мира.

Основная часть. Для упрощения и ускорения работы были разработаны алгоритмы процедурной генерации. В целом, процедурные методы представляют собой сегменты кода или алгоритмы, позволяющие определить некоторые характеристики сгенерированной компьютером модели или эффекта [1]. При процедурном подходе вместо явного указания и сохранения всех сложных деталей сцены они абстрагируются в функцию или алгоритм, и уже вызывается эта процедура, когда и где это необходимо. По итогу уменьшается хранимая информация, поскольку детали больше не указываются явно в файлах хранения, а подразумеваются в процедуре, при этом временные требования к уточнению деталей перекладываются с программиста на компьютер. Это позволяет создавать модели с несколькими разрешениями и текстуры, которые мы можем оценить с требуемым разрешением. Также дается возможность параметрического управления, позволяющая нам присвоить параметру значимую концепцию (например, число, которое делает горы более грубыми или плавными). Этот параметрический контроль освобождает пользователя от низкоуровневого контроля и детализации.

Перед тем как подробно останавливаться на различных методах, кратко рассмотрим применение процедурной генерации, что с помощью этих алгоритмов можно создавать и как это используется в последствии. Основным видом контента является генерация различных игровых уровней, миров, карт, топологий, головоломок и лабиринтов. Примером может послужить игра Minecraft, в которой каждый создаваемый игровой мир получается при помощи алгоритмов процедурной генерации, в частности шума Перлина, о котором поговорим подробнее чуть позже. По итогу локации отличаются от ранее сгенерированных, что заставляет игрока погрузиться в их изучении еще глубже, а это служит тому, что игра может просуществовать десятилетиями. Пример такой генерации представлен на рисунке 1. Помимо уровней и карт также требуется генерировать различные объекты (деревья, облака, жидкости и т.д.). С течением времени развития алгоритмов были разработаны методы моделирования на более высоком уровне, которые позволяют абстрагировать модель, кодировать классы объектов и обеспечивают более высокий уровень контроля и спецификации моделей. Некоторые передовые методы геометрического моделирования, основанные на грамматиках по типу графтал и L-систем,

позволяют задавать несколько параметров для моделирования сложных природных объектов, таких как деревья и растения [1].

Рисунок 1 – Результат выполнения нескольких алгоритмов процедурной генерации, представленной в игре Minecraft

Системы частиц представляют собой еще один процедурный метод моделирования, который позволяет абстрагировать и специфицировать объекты и их анимацию. В этих системах объект представляется большой коллекцией простых геометрических частиц, которые стохастически меняются со временем. Таким образом, системы частиц используют базу данных геометрических примитивов для представления естественных объектов, но анимация, местоположение, рождение и исчезновение частиц, представляющих объект, контролируются алгоритмически. Как и в случае с другими процедурными методами моделирования, системы частиц позволяют разработчику указывать и контролировать эту большую базу данных с помощью нескольких параметров. Недостатком этого подхода является то, что он требует расширения базы данных для обеспечения реалистичного представления объектов [1].

Рассмотрим несколько алгоритмов, которые можно реализовать на языке программирования Python. Использоваться они будут для генерации примитивного трехмерного объекта – прямоугольного параллелепипеда. Процедурная генерация сеток. В компьютерной графике сетка (mesh) представляет собой расположение точек, линий и поверхностей, которые очерчивают форму и структуру 3D-объекта или поверхности. Поэтому быстрая генерация такой структуры представляет важную ценность. Для создания программы, реализующей процедурную генерацию сетки параллелепипеда, можно использовать следующий подход. Мы будем генерировать вершины и индексы для треугольников, которые составляют грани параллелепипеда. Важно отметить, что каждая грань параллелепипеда представляет собой прямоугольник, который можно разбить на два треугольника для формирования сетки.

Для простоты рассмотрим параллелепипед с центром в начале координат и заданными полуосями по каждому из измерений (половина длины по x, y и z). Тогда вершины параллелепипеда можно определить через эти значения. После этого мы находим индексы вершин для каждого из 12 треугольников, составляющих 6 граней параллелепипеда. Все это выполняет функция generate_parallelepiped_mesh(dx, dy, dz), в которую передаются размеры по координатам, а на выходе получаем массив вершин сетки vertices и треугольников triangles.

```
    def generate_parallelepiped_mesh(dx, dy, dz):
  vertices = np.array([
    [-dx, -dy, -dz],
    [dx, -dy, -dz],
    [dx, dy, -dz],
    [-dx, dy, -dz],
    [-dx, -dy, dz],
    [dx, -dy, dz],
```

```
        [dx, dy, dz],
        [-dx, dy, dz]
    ])
    triangles = np.array([
        [0, 1, 2], [2, 3, 0],
        [4, 6, 5], [6, 4, 7],
        [0, 4, 5], [5, 1, 0],
        [3, 2, 6], [6, 7, 3],
        [0, 3, 7], [7, 4, 0],
        [1, 5, 6], [6, 2, 1]
    ])
    return vertices, triangles
```

Чтобы визуализировать сгенерированный mesh параллелепипеда в 3D можно использовать библиотеку matplotlib в Python. Этот подход позволяет легко рендерить 3D-объекты и работает хорошо для простых демонстраций и тестирования. За рендер объектов отвечает процедура plot_objects(objects), в которую подается список объектов, состоящих из vertices, triangles и цвета RGBH.

```
    def plot_objects(objects):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    for vertices, triangles, color in objects:
        # Построение граней для каждого объекта
        for tri in triangles:
            triangle = vertices[tri]
            poly = Poly3DCollection([triangle])
            poly.set_edgecolor('k')
            poly.set_facecolor(color)  # Цвет задается для каждого объекта
            ax.add_collection3d(poly)

    # Автоматическое масштабирование для отображения всех объектов
    all_vertices = np.vstack([obj[0] for obj in objects])
    ax.auto_scale_xyz(all_vertices[:, 0], all_vertices[:, 1], all_vertices[:, 2])
    plt.show()
```

Результат работы представлен на следующем рисунке 2:

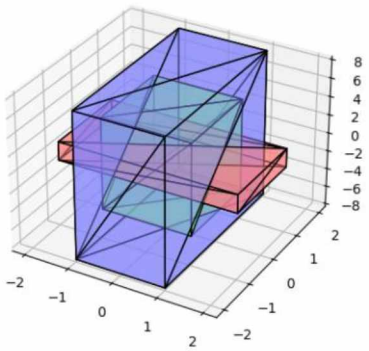

Рисунок 2 – Результат процедурной генерации сеток нескольких параллелепипедов

Шум Перлина. После того, как был сформирован объект, следующей задачей становится размещение его на рабочем трехмерном поле. Для решения этой задачи существует множество алгоритмов, одним из них можно считать шум Перлина. Шум

Перлина — это тип градиентного шума, разработанный Кеном Перлином. Его можно использовать для процедурной генерации таких объектов, как текстуры и рельеф местности без их ручного создания художником или дизайнером [4].

В отличие от обычного белого шума, у Перлина есть плавные «переходы» между случайными локальными максимумами и минимумами.

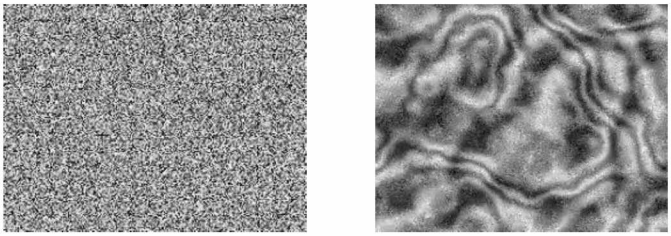

Рисунок 3 - Белый шум (слева) и шум Перлина (справа)

Алгоритм принимает в качестве входных данных определенное количество параметров с плавающей запятой (в зависимости от размера) и возвращает значение в определенном диапазоне. Входными данными чаще всего являются:

1 Octaves – количество кривых Перлина, отвечающих за неоднородность шума.

2 Amp – коэффициент, отвечающий за итоговую высоту координаты y.

3 Period – это периодичность пиков кривой Перлина. При ее увеличении поверхность становится более гладкой.

4 Seed – число, которое однозначно описывает генерацию.

Возвращаемая величина является высотой, каждая из которых отображается своим цветом [2]. Для работы с шумом Перлина в Python существует специальный модуль perlin-noise, позволяющий облегчить работу с алгоритмом. Следующий код демонстрирует генерацию шума Перлина со случайно выбранным seed в диапазоне от 1000 до 999999 с шагом 50:

```
def gen_seed():
    return random.randrange(1000, 999999, 50)
# генерация основного шума и параметризация
seed = gen_seed()
noise = PerlinNoise(octaves=2, seed = seed)
amp = 6
period = 24
map_width = 50
#генерация матрицы для представления ландшафта
land = [[0 for i in range(map_width)] for i in range(map_width)]
for position in range(map_width**2):
   # вычисление высоты y в координатах (x, z)
   x = floor(position / map_width)
   z = floor(position % map_width)
   y = floor(noise([x/period, z/period])*amp)
   land[int(x)][int(z)] = int(y)
   if int(y) > 1:
      print(int(x), int(y), int(z))
```

Результатом работы является следующее изображение (рисунок 4):

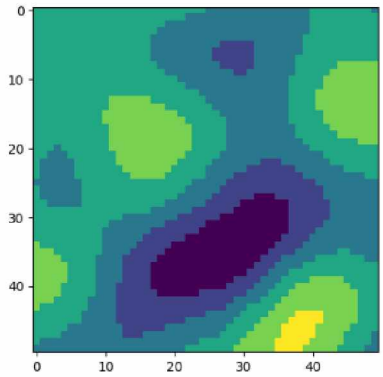

Рисунок 4 – Сгенерированный шум Перлина

На нем четко виднеются различные цветовые зоны, которые отображают высоты. В данном примере желтым отображается высота 2, а темно-синим являются точки с высотой -3. Уже по полученному шуму можно визуализировать наши объекты. Для этого вновь воспользуемся модулем matplotlib в Python. В данном случае мы циклами проходимся по существующему шуму и выбираем координаты, где высота больше либо равна нулю. На их местах генерируются параллелепипеды, их высота определяется на основе шума:

```
cube_size = 1
for i in range(map_width):
    for j in range(map_width):
        # Выбираем только "белые поля", где значение шума Перлина высоко
        if land[int(i)][int(j)] >= 0:
            # Координаты нижней левой вершины куба
            x, y, z = i, 0, j
            # Добавляем куб
            ax.bar3d(x, y, z, cube_size, land[int(x)][int(z)], cube_size)
```

Результат генерации объектов методом шума Перлина (рисунок 5):

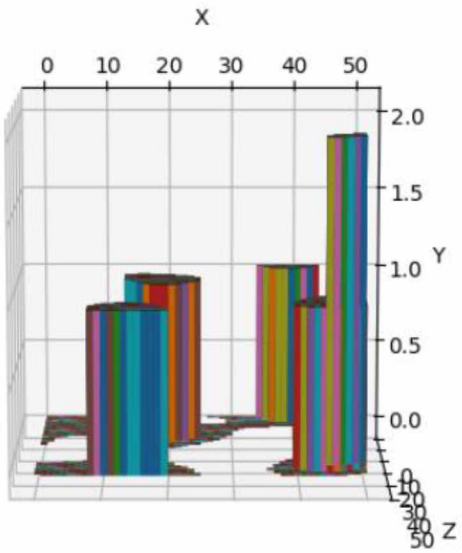

Рисунок 5 – Результат процедурной генерации нескольких параллелепипедов шумом Перлина

Заключение. Используя описанные алгоритмы, можно создавать различные объекты, определяя только их размеры, местоположение и их количество задаются во время процедурной генерации.

Список литературы

1. David S. Ebert, Ken Perlin, Steven Woorley, Darwyn Peachey,F.KentonMusgrave "Texturing & Modeling: A Procedural Approach"

2. Простая процедурная генерация мира, или Шумы Перлина на Python https://habr.com/ru/companies/selectel/articles/731506/ (дата обращения 15.03.2024)

3. Шум Перлина: процедурный алгоритм генерации https://rtouti.github.io/graphics/perlin-noise-algorithm (дата обращения 20.03.2024)

4. William L. Raffe, Fabio Zambetta, and Xiaodong Li "A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms"

5. Генерация сеток на Python https://www.geeksforgeeks.org/generating-meshes-in-python/ (дата обращения 24.03.2024)

UDC 004.041, 004.925.84

PROCEDURAL GENERATION OF THREE-DIMENSIONAL PRIMITIVE OBJECTS

Seleznev A.S.

Ryazan State Radio Engineering University named after V.F. Utkin, Ryazan, Russian Federation

Bakulev A.V.- Cand. of Sci., associate professor, associate professor of the department of of CAD CS
Bakuleva M.A. - Cand. of Sci., associate professor, associate professor of the department of of CAD CS

Abstract. The article discusses the use of various methods of procedural generation of three-dimens ional objects that are currently used, as well as the implementation of some algorithms in Python.

Keywords: procedural generation, 3D modeling, Python, program