Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**КУРСОВАЯ РАБОТА**

Исследование методов оптимизации алгоритма сканирования ландшафта для построения карт высот

по дисциплине «Операционные системы»

Выполнили

студент гр. 5130901/20103

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кудрин Ф.Г.

(подпись)

студент гр. 5130901/20102

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Вагнер А.А.

(подпись)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Куляшова З.В.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург   
2024

## ВВЕДЕНИЕ

Повышение разрешения данных о скане местности представляет собой актуальную задачу в геоинформатике, робототехнике, картографии и других областях, связанных с обработкой пространственной информации. Исходные данные таких сканов часто представлены в виде матрицы, где каждый элемент указывает расстояние до поверхности местности в определённой точке. Однако, в силу ограничений оборудования, таких как разрешение сканирующих устройств, точность измерений и плотность точек, данные могут быть недостаточно детализированными для решения сложных задач. Поэтому повышение разрешения становится важным шагом для улучшения качества визуализации и анализа местности.

В данном исследовании рассматриваются подходы к повышению разрешения данных с использованием математических и вычислительных методов. Основная цель таких методов — увеличение плотности точек данных при сохранении (или улучшении) точности и качества интерпретации местности. Среди множества существующих методов в данной работе рассматриваются два подхода:

1. **Интерполяционный подход**, основанный на использовании математических полиномов для вычисления промежуточных значений между точками исходной матрицы. Этот метод строится на анализе локальных зависимостей между соседними точками и позволяет аппроксимировать значения на более мелкой сетке, увеличивая плотность данных.
2. **Масштабирование**, подход который заимствован из обработки изображений. Этот подход предполагает применение современных методов увеличения разрешения, таких как алгоритмы интерполяции и фильтрации, позволяющие обработать матрицу данных аналогично цифровому изображению. Этот метод способен учитывать пространственные особенности данных и эффективно восстанавливать детали.

Оба метода имеют свои преимущества и ограничения, а также различную область применимости. Их анализ и сравнение позволяют определить наиболее эффективный подход для повышения разрешения в зависимости от характеристик исходных данных и целевых задач. Выбор подхода также зависит от требуемой вычислительной сложности и допустимой точности результата.

В результате применения описанных методов можно добиться значительного улучшения качества данных, что открывает новые возможности для их использования в прикладных задачах, таких как построение детализированных 3D-моделей, точное определение рельефа местности, а также разработка навигационных систем.

Для достижения цели исследования, связанной с повышением разрешения данных о скане местности, планируется выполнение следующих этапов:

1. **Обзор существующих методов повышения разрешения данных**  
   На этом этапе будет проведён анализ литературы, посвящённой подходам к интерполяции и увеличению разрешения матричных данных. Особое внимание будет уделено математическим методам интерполяции (например, на основе полиномов) и подходам, заимствованным из обработки изображений, включая современные алгоритмы апскейлинга.
2. **Выбор алгоритмов для реализации**

На основе проведённого анализа будут выбраны два алгоритма, представляющие разные подходы к повышению разрешения:

* + Интерполяционный метод (например, метод полиномов Лагранжа, сплайнов или других функций).
  + Метод апскейла изображений (например, билинейная интерполяция, бикубическая интерполяция, или современные методы, такие как суперрезолюция на основе нейронных сетей).  
    Цель этого этапа — определить наиболее подходящие алгоритмы для дальнейшей разработки.

1. **Разработка кода для реализации выбранных алгоритмов**
   * Реализация интерполяционного алгоритма для повышения разрешения матричных данных.
   * Разработка кода для апскейла данных, используя подходы, заимствованные из обработки изображений.  
     При этом особое внимание будет уделено корректности и оптимизации кода, чтобы обеспечить достаточную производительность.
2. **Тестирование алгоритмов на контрольных данных**
   * Подготовка тестовых наборов данных, представляющих собой матрицы с данными о местности (реальные или сгенерированные).
   * Проведение тестирования разработанных алгоритмов на этих данных для оценки их корректности, вычислительной сложности и качества получаемых результатов.
3. **Сравнительный анализ результатов работы алгоритмов**
   * Оценка качества повышения разрешения для обоих методов. Метрики оценки могут включать визуальное соответствие полученных данных исходной местности, точность восстановления на контрольных точках и числовые характеристики, такие как среднеквадратичная ошибка.
   * Сравнение вычислительных затрат обоих подходов для оценки их эффективности.
   * Выводы о сильных и слабых сторонах каждого метода.
4. **Оформление результатов работы**  
   Подготовка итогового отчёта, включающего описание разработанных методов, результаты тестирования, сравнительный анализ и рекомендации по применению.

Unity был выбран в качестве среды для проведения тестирования благодаря ряду его преимуществ, которые делают его идеальной платформой для работы с данными о местности и их визуализации:

1. **Мощный физический движок**  
   Unity оснащён современным физическим движком, который позволяет точно моделировать взаимодействие объектов в трёхмерном пространстве. Это особенно важно при тестировании данных о местности, поскольку он предоставляет возможность визуализировать результаты повышения разрешения в физически корректной среде. Например, можно оценить, как изменения данных влияют на точность рельефа или траекторию движения объектов по местности.
2. **Удобство работы с 3D-геометрией и рельефом**  
   Unity имеет встроенные инструменты для работы с высотными картами (heightmaps) и рельефом, что позволяет легко импортировать матрицы данных, преобразовывать их в визуальные 3D-модели и проводить анализ. Это делает его удобным инструментом для тестирования результатов повышения разрешения данных о местности.
3. **Гибкость и расширяемость**  
   Unity поддерживает написание скриптов на C#, что даёт возможность реализовать пользовательские алгоритмы и интегрировать их непосредственно в тестовую среду. Разработанные алгоритмы повышения разрешения можно протестировать в Unity, визуализировать их результаты, а также оценить поведение различных объектов на обработанной местности.

## ХОД РАБОТЫ

### Обзор существующих методов повышения разрешения данных

**Аналитический метод интерполяции**

В качестве аналитического метода интерполяции был выбран Интерполяционный полином Ньютона 4 степени, была выдвинута теория, что алгоритм построения полиномов больших степеней можно будет использовать для дальнейшего повышения точности интерполяции в случае, если данный алгоритм окажется наиболее оптимальным.

Оценка погрешности на основе производной не представляется возможной, поэтому для измерения точности интерполированных значений будет также использоваться среднеквадратичное отклонение.

Формула полинома приведена ниже

Так как специфика задачи не подразумевает различные интервалы между точками измерения, следует использовать интерполяционный полином Ньютона для равноотстоящих узлов. Пусть h – постоянный шаг измерений, тогда .

Алгоритм обработки.

Соберём выборку данных для интерполяции, матрица Х\*Х точек, эта матрица обрабатывается алгоритмом интерполяции, который по наборам точек из каждого ряда строит полином и интерполирует точки между уже имеющимися, таким образом ширина матрицы возрастает вдвое. Далее подобный алгоритм применяется к столбцам матрицы, полученным на предыдущем шаге. Таким образом результирующая матрица имеет разрешение в 4 раза больше изначальной.

Алгоритм тестирования

Для тестирования требуется набор более точных данных, для этого проведём измерение с разрешением равным разрешению полученных методом интерполяции данных и найдём среднеквадратичное отклонение. Следует отметить, что данная метрика не подлежит качественному анализу, так как измерения расстояний на виртуальном физическом движке невозможно достоверно перенести на реальное пространство. В конце работы следует провести сравнительный анализ данных тестов различных методов.

**Алгоритмы повышения масштаба изображения**

Алгоритмы повышения масштабов изображения используются для увеличения разрешения изображений, сохраняя при этом их качество и четкость. Существуют различные методы интерполяции и алгоритмы, каждый из которых имеет свои особенности, преимущества и недостатки. Вот некоторые из наиболее распространенных подходов:

**1. Билинейная интерполяция**

Билинейная интерполяция учитывает четыре ближайших пикселя (по два по горизонтали и по вертикали) для вычисления нового значения пикселя. Она применяет линейную интерполяцию сначала по одной оси, а затем по другой.

Плюсы:

- Простота реализации.

- Умеренное качество при увеличении.

Минусы:

- Мягкие и размытые края.

- Может терять детали при значительном увеличении.

**2. Бикубическая интерполяция**:

Бикубическая интерполяция использует 16 ближайших пикселей (по 4 по горизонтали и вертикали) для вычисления нового значения. Она обеспечивает более гладкие переходы и лучше сохраняет детали.

Плюсы:

- Лучше качество по сравнению с билинейной интерполяцией.

- Сохраняет больше деталей и текстур.

Минусы:

- Более сложная в реализации.

- Меньшая скорость по сравнению с билинейной интерполяцией.

**3. Алгоритмы на основе сверток:**

Эти алгоритмы используют свертки для обработки изображения, применяя различные фильтры для увеличения. Например, можно использовать фильтры размытия или резкости.

Плюсы:

- Гибкость в выборе фильтров.

- Можно улучшить конкретные аспекты изображения.

Минусы:

- Сложность в настройке параметров фильтров.

- Может требовать значительных вычислительных ресурсов.

**4. Scale2x и Scale3x**:

Эти алгоритмы разработаны для увеличения пиксельной графики, обеспечивая четкость и контрастность. Scale2x увеличивает изображение в 2 раза, а Scale3x — в 3 раза.

Плюсы:

- Упрощает улучшение качества для пиксельной графики.

- Подходит для ретро-игр и эмуляторов.

Минусы:

- Могут возникать артефакты.

- Не всегда подходят для изображений со сложными текстурами.

В данной работе воспользуемся последними представленными алгоритмами, так как исходное изображение, которое будет преобразовываться должно сохранить пиксельный формат.

### Разработка кода для реализации выбранных алгоритмов

В первую очередь, необходимо реализовать модуль сканера для получения информации об объектах на сцене, а также реализовать модуль создания и вывода изображения.

Для этого была создана новая Сцена в Unity? А также были описаны следующие модули:

Изображение выглядит как Мультимедийное программное обеспечение, программное обеспечение, Графическое программное обеспечение, мультимедиа

Автоматически созданное описание

Рис 1. – Тестовая сцена с реализованными модулями сканера и монитором

Листинг 1. – Описание модуля **ScanerTest:**

**using** System**.**Runtime**.**CompilerServices**;**

**using** Unity**.**Hierarchy**;**

**using** Unity**.**VisualScripting**;**

**using** UnityEngine**;**

**public** class ScanerTest **:** MonoBehaviour

**{**

**[**SerializeField**]** **private** TextureTest \_screen**;**

**[**Range**(**1**,** 100000**)]**

**[**SerializeField**]** **private** float \_depth **=** 50**;**

**[**Range **(**1**,**10**)]**

**[**SerializeField**]private** float \_scanerSize**;**

**[**SerializeField**]** **private** float \_updateTime**;**

**[**Range**(**1**,** 1000**)]**

**[**SerializeField**]** **private** int \_resolution**;**

**[**Range**(**1**,** 1000**)]**

**[**SerializeField**]** **private** int \_colorStep**;**

**[**SerializeField**]** **private** float \_minHorizontalAngle**;**

**[**SerializeField**]** **private** float \_maxHorizontalAngle**;**

**[**SerializeField**]** **private** float \_minVerticalAngle**;**

**[**SerializeField**]** **private** float \_maxVerticalAngle**;**

**private** Ray**[,]** \_rays**;**

**private** float \_scanTime**;**

**private** float \_lenseRadius**;**

**private** float**[,]** \_distances**;**

**public** float**[,]** Distances **{** **get** **{** **return** \_distances**;** **}** **}**

**public** float Depth **{** **get** **{** **return** \_depth**;** **}** **}**

**private** void Start**()**

**{**

//if (\_screen != null) \_resolution = \_screen.Resolution;

\_rays **=** **new** Ray**[**\_resolution**,** \_resolution**];**

\_distances **=** **new** float**[**\_resolution**,** \_resolution**];**

\_lenseRadius **=** \_scanerSize**/**Mathf**.**Sqrt**(**2**);**

InitializedRays**();**

**if** **(**\_screen **!=** **null)** CheckRays**();**

\_scanTime **=** \_updateTime**;**

**}**

**private** void Update**()**

**{**

\_scanTime **-=** Time**.**deltaTime**;**

**if** **(**\_scanTime **<** 0**)**

**{**

**if** **(**\_screen **!=** **null)** CheckRays**();**

\_scanTime **=** \_updateTime**;**

**}**

**}**

**private** void InitializedRays**()**

**{**

float horizontalStep **=** **(**\_maxHorizontalAngle **-** \_minHorizontalAngle**)** **/** **(**\_resolution**);**

float verticalStep **=** **(**\_maxVerticalAngle **-** \_minVerticalAngle**)** **/** **(**\_resolution**);**

**for** **(**int x **=** 0**;** x **<** \_resolution**;** x**++)**

**{**

**for** **(**int y **=** 0**;** y **<** \_resolution**;** y**++)**

**{**

float horizontalAngle **=** \_minHorizontalAngle **+** x **\***horizontalStep**;**

float verticalAngle **=** \_minVerticalAngle **+** y **\*** verticalStep**;**

Vector3 direction **=** CalculateDirection**(**horizontalAngle**,** verticalAngle**);**

\_rays**[**x**,**y**]** **=** **new** Ray**(**transform**.**position**,** direction**);**

**}**

**}**

**}**

**private** void CheckRays**()**

**{**

**for** **(**int x **=** 0**;** x **<** \_resolution**;** x**++)**

**{**

**for** **(**int y **=** 0**;** y **<** \_resolution**;** y**++)**

**{**

**if** **(**Physics**.**Raycast**(**\_rays**[**x**,** y**],** **out** RaycastHit hit**,** \_depth**))**

**{**

\_distances**[**x**,**y**]** **=** hit**.**distance**;**

**}**

**else**

**{**

\_distances**[**x**,** y**]** **=** \_depth**+**1**;**

**}**

**}**

**}**

\_screen**.**IsChange **=** **true;**

**}**

Vector3 CalculateDirection**(**float horizontalAngle**,** float verticalAngle**)**

**{**

float hRad **=** Mathf**.**Deg2Rad **\*** horizontalAngle**;**

float vRad **=** Mathf**.**Deg2Rad **\*** verticalAngle**;**

Vector3 direction **=** **new** Vector3**(**Mathf**.**Sin**(**hRad**),** Mathf**.**Sin**(**vRad**),** Mathf**.**Cos**(**hRad**)** **\*** Mathf**.**Cos**(**vRad**));**

direction**.**Normalize**();**

**return** direction**;**

**}**

**}**

Листинг 2. - Описание модуля **TextureTest:**

**using** Unity**.**VisualScripting**;**

**using** UnityEngine**;**

**public** enum Source

**{**

Interpolator**,**

Scaner

**}**

**public** class TextureTest **:** MonoBehaviour

**{**

**[**SerializeField**]** **private** Texture2D \_texture**;**

**[**Range**(**10**,** 1000**)]**

**[**SerializeField**]** **private** int \_resolution **=** 10**;**

**[**SerializeField**]** **private** FilterMode \_filterMode **=** FilterMode**.**Point**;**

**[**SerializeField**]** **private** ScanerTest \_scaner**;**

**[**SerializeField**]** **public** Source \_source**;**

**[**SerializeField**]** **private** Interpolator \_interpolator**;**

**[**Header**(**"Color Settings"**)]**

**public** Color minColor **=** Color**.**green**;**

**public** Color maxColor **=** Color**.**red**;**

**private** Color**[,]** \_pixels**;**

**public** bool IsChange **{** **get;** **set;** **}**

**public** int Resolution **{** **get** **{** **return** \_resolution**;** **}** **}**

**public** Color**[,]** Pixels **{** **get** **{** **return** \_pixels**;** **}** **}**

**private** float minDistance **=** 0.0f**;**

**private** float maxDistance **=** **-**1**;**

**private** void Start**()**

**{**

maxDistance **=** \_scaner**.**Depth **\*** 1.1f**;**

\_texture **=** **new** Texture2D**(**\_resolution**,** \_resolution**);**

\_texture**.**filterMode **=** \_filterMode**;**

GetComponent**<**Renderer**>().**material**.**mainTexture **=** \_texture**;**

\_pixels **=** **new** Color**[**\_resolution**,** \_resolution**];**

InitializedPixels**();**

RedrawTexture**();**

**}**

**private** void RedrawTexture**()**

**{**

**for** **(**int x **=** 0**;** x **<** \_resolution**;** x**++)**

**{**

**for** **(**int y **=** 0**;** y **<** \_resolution**;** y**++)**

**{**

\_texture**.**SetPixel**(**x**,** y**,** \_pixels**[**x**,** y**]);**

**}**

**}**

\_texture**.**Apply**();**

IsChange **=** **false;**

**}**

**private** void InitializedPixels**()**

**{**

**for** **(**int x **=** 0**;** x **<** \_resolution**;** x**++)**

**{**

**for** **(**int y **=** 0**;** y **<** \_resolution**;** y**++)**

**{**

\_pixels**[**x**,** y**]** **=** Color**.**black**;**

**}**

**}**

**}**

**private** void ChangeColor**()**

**{**

**if** **(**\_source **==** Source**.**Scaner**)**

**{**

**for** **(**int i **=** 0**;** i **<** \_resolution**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** \_resolution**;** j**++)**

**{**

float distance **=** \_scaner**.**Distances**[**i**,** j**];**

**if** **(**distance **<=** \_scaner**.**Depth**)** **{**

float t **=** **(**distance **-** minDistance**)** **/** **(**maxDistance **-** minDistance**);**

**if** **(**t **>** 1**)** t **=** 1**;**

\_pixels**[**i**,** j**]** **=** Color**.**Lerp**(**minColor**,** maxColor**,** t**);**

**}** **else** \_pixels**[**i**,** j**]** **=** Color**.**black**;**

**}**

**}**

**}**

**else**

**{**

**for** **(**int i **=** 0**;** i **<** \_resolution**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** \_resolution**;** j**++)**

**{**

float distance **=** \_interpolator**.**Distances**[**i**,** j**];**

**if** **(**distance **<=** \_scaner**.**Depth**)**

**{**

float t **=** **(**distance **-** minDistance**)** **/** **(**maxDistance **-** minDistance**);**

**if** **(**t **>** 1**)** t **=** 1**;**

\_pixels**[**i**,** j**]** **=** Color**.**Lerp**(**minColor**,** maxColor**,** t**);**

**}**

**else** \_pixels**[**i**,** j**]** **=** Color**.**black**;**

**}**

**}**

**}**

**}**

**private** void LateUpdate**()**

**{**

**if** **(**IsChange **&&** \_scaner **!=** **null)**

**{**

ChangeColor**();**

RedrawTexture**();**

**}**

**}**

**}**

Для модулей ScanerTest и TextureTest поле \_resolution, определяет размер квадратной матрицы получаемых и обрабатываемых точек. Размерность данного поля задается в диапазоне от [10, 1000], так как большие значения вызывают сильную нагрузку системы и прерывание выполнения приложения, а меньшее количество будет недостаточно информативным.