САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И СИСТЕМ

Плаксин Вадим Александрович

Магистерская диссертация

Восстановление трехмерной модели строений археологического памятника по материалам камеральных исследований

Направление 020402

Фундаментальная информатика и информационные технологии

Магистерская программа Автоматизация научных исследований

Научный руководитель,

Доктор технических наук,

профессор

Дегтярев А. Б.

Санкт-Петербург

2019

Содержание

Введение

В настоящее время в поисках исторических архитектурных сооружений археологами ведется большое число исследований по всему миру. Среди огромного числа методов, используемых в этих исследованиях, важное место занимает обязательная визуальная опись каждого найденного строения и его структуры. Эта опись в основном производится в виде специальных планов и абрисов (абрис – контур, набросок, очертание предмета, линия, показывающая форму какого-то объекта). В дальнейшем полученные результаты могут подвергаться трехмерной реконструкции для облегчения исследовательской деятельности. Но это трудоемкий процесс, который требует от исследователей специальных навыков работы с компьютером и программным обеспечением.

Автоматическое компьютерное моделирование исторических объектов в виде трехмерной визуализации на основе двумерного изображения позволило бы если не полностью избавить археологов от этой задачи, то хотя бы упростить ее решение, предоставляя некоторую предобработанную трехмерную модель, которую можно мануально довести до ожидаемого результата.

Также археологам часто приходится иметь дело с не полностью сохранившимися архитектурными объектами исторической ценности (например, разрушенная каменная стена). В таком случае специалистам приходится долгое время вручную создавать модели не только сохранившихся частей, но и к тому же моделировать утраченные со временем. Таким образом предыдущая задача расширяется требованием попытаться восстановить недостающие составляющие архитектурного памятника по имеющимся данным.

Постановка задачи

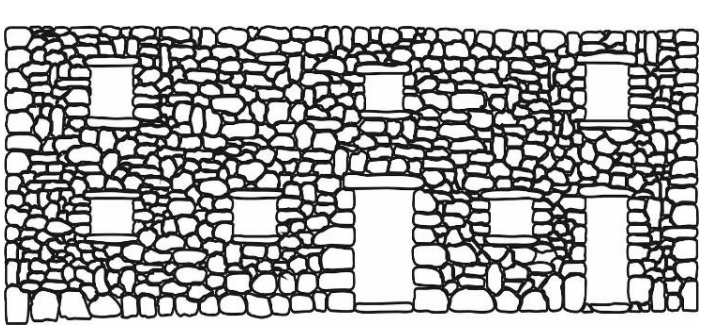
Целью работы является создание программного обеспечения, способного решить выявленные ранее задачи трехмерного моделирования архитектурных сооружений на примере каменной стены по двумерным изображениям с возможностью достраивания недостающих частей.

Для этого можно поставить следующие задачи:

1. Трехмерная визуализация строения по двумерному абрису (может быть разбита на следующие подзадачи)
   - 1.1 Выделение из исходного изображения малых составляющих (абрисы отдельных камней).
   - 1.2 Построение трехмерной модели отдельного камня по его абрису.
   - 1.3 Построение трехмерной модели стены по полученным в предыдущей задаче моделям отдельных камней и информации из общего абриса стены (расположение камней в стене)
2. Восстановление недостающих частей архитектурного сооружения (может быть поставлена различным образом в зависимости от имеющихся данных)
   - 2.1 Восстановление трехмерной модели сооружения по малой информации (модели отдельных камней и некоторые параметры стены такие, как ширина, высота и толщина стены, траектория стены и др.)
   - 2.2 Восстановление трехмерной модели стены с помощью двумерной генерации недостающих частей абриса и далее с помощью решения задачи 1 моделирование стены по вновь сгенерированному абрису.

Обзор литературы

В целом автоматические процессы в трехмерном моделировании архитектуры исторических памятников не сильно развиты в текущее время. На данный момент для трехмерной визуализации исследователи в ручном режиме используют популярные 3D редакторы такие, как 3ds Max [1], Blender [2], Cinema 4D и др. В этих программных пакетах уже есть базовый функционал, позволяющий по абрису восстановить внешний вид стены. Например, на Рис.1 изображена прорись кладки стены, а на Рис.2 результат работы функции Displace в Houdini [3], которая переводит карту глубины в рельефную текстуру стены.



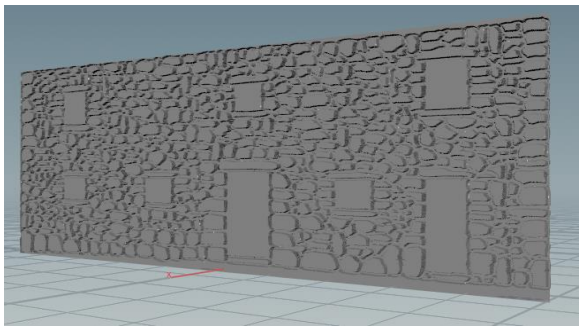


Рис.1, Рис.2 – абрис и результат работы функции Displace

Однако при таком подходе мы получаем рельефную стену, как один большой 3D объект, который нельзя разделить на несколько отдельных камней, что лишает пользователя возможности смоделировать физическое отношение между камнями, а также ограничивает возможность исключительно локально редактировать получившуюся модель. Например, так невозможно смоделировать разрушение стены под физическим воздействием.

Также существуют некоторые решения [4], которые позволяют избежать проблем, описанных выше:

1. Реконструкция, основанная на наборе фотографий или видеозаписей сделанных с разных углов
2. Реконструкция, основанная на наборе данных, полученных с помощью сканирования поверхности без извлечения

исторического объекта из места его расположения.

Но эти способы имеют ограничение в виде требования огромного количества входных данных, что не всегда возможно предоставить.

Таким образом, задачи восстановления трехмерной модели архитектурного сооружения по ограниченным данным (абрис стены), как никогда актуальны.

Глава 1. Построение трехмерной модели строения по двумерному абрису

1.1 Предобработка изображения

Исходными данными для задачи построения трехмерной модели по абрису будет являться черно-белое изображение абриса (если изображение не черно-белое, то сначала бинаризуем его так, чтобы можно было выделить отдельные камни). Первым делом нужно предобработать исходное изображение: с помощью морфологических операций [5]

1. открытие - последовательное применение операций расширение и сужение с одним и тем же структурным элементом
2. закрытие - последовательное применение операций сужение и расширение с одним и тем же структурным элементом

избавляемся от шумов в виде “перец и соль” в абрисе.

Далее с помощью детектора границ Кэнни мы можем определить контуры камней и выделить среди них замкнутые границы. Кроме самих границ будем сохранять их иерархию (от внешних границ к внутренним), чтобы вычислить и не учитывать в дальнейшем процессе внешние границы, которые будут обтекать всю стену целиком, а не отдельный камень.

Причем, вместе с границами мы вычисляем положение камня в стене, через положение геометрического центра огибающего его прямоугольника:

$$x_{center} = \frac{x_{max} - x_{min}}{2};\ y_{center} = \frac{y_{max} - y_{min}}{2}$$, где

x_{max} – наибольшая абсцисса среди всех точек границы

x_{min} – наименьшая абсцисса среди всех точек границы

y_{max} – наибольшая ордината среди всех точек границы

y_{min} – наименьшая ордината среди всех точек границы

$(x_{center}\ ;\ y_{center})$ – положение камня в стене

1.2 Построение облака точек камня

В итоге, после предобработки изображения мы получаем границы каждого камня (Рис.3) по отдельности в виде массива точек, образующих эту границу. Теперь построим трехмерную модель по абрису. Для этого нам понадобится еще один контур, который будет определять вид камня сбоку. Этот контур может быть выбран произвольно (Рис.4).

Рис.3 Рис.4 – контуры, по которым восстанавливается модель камня

Идея построения модели заключается в следующем: один из контуров будем называть шаблонным, второй – статическим (контур не будет изменен в результате работы алгоритма). Будем откладывать новые контуры с некоторым шагом вдоль оси перпендикулярной плоскости новых контуров и параллельной плоскости статического. При этом это движение должно удовлетворять следующим условиям:

1. Новые контуры должны быть подобны шаблонному контуру (то есть новые контуры – есть шаблонный контур с измененным масштабом)
2. Точки пересечения новых контуров с плоскостью статического контура должны принадлежать статическому контуру.

Рассмотрим процесс построения одного контура подобного шаблонному:

1. Положим, что статический контур лежит в плоскости Oxy, а шаблонный - в плоскости Oyz. Значит пересечение плоскостей двух контуров будет параллельно оси Oy.

2. Зафиксируем какой-нибудь x_{fix}. Найдем максимальное y_{max}^{static} и минимальное y_{min}^{static} среди точек статического контура с выбранным x_{fix}.

$$\begin{cases} y_{max}^{static} = \max\{y \mid (x_{fix}, y) \in StaticContour\} \\ y_{min}^{static} = \min\{y \mid (x_{fix}, y) \in StaticContour\} \end{cases}$$

где StaticContour – множество точек статического контура.

3. Найдем центр шаблонного контура с помощью вычисления моментов этого контура:

$$m(p,q) = \sum z^p y^q|_{(z,y)\in TemplateContour}$$

$$\begin{cases} z_{center}^{template} = \dfrac{m(1,0)}{m(0,0)} \\ y_{center}^{template} = \dfrac{m(0,1)}{m(0,0)} \end{cases}$$

где TemplateContour – множество точек шаблонного контура

4. Проделаем с шаблонным контуром те же вычисления, что и со статическим на 2 шаге алгоритма, только вместо x_{fix} возьмем $z_{center}^{template}$:

$$\begin{cases} y_{max}^{template} = \max\{y \mid (z_{center}^{template}, y) \in TemplateContour\} \\ y_{min}^{template} = \min\{y \mid (z_{center}^{template}, y) \in TemplateContour\} \end{cases}$$

Если $y_{max}^{template} = y_{min}^{template}$, то такой шаблонный контур неверно размечен и его можно не рассматривать.

5. Теперь нам остается найти масштаб (scale) и перемещение (displacement), с помощью которых мы сможем разместить точки шаблонного контура на плоскости x = x_{fix} так, чтобы отрезок с концами в точках ($z_{center}^{template}$; $y_{min}^{template}$) и ($z_{center}^{template}$; $y_{max}^{template}$) перешел бы в отрезок с концами в точках (x_{fix} ; y_{min}^{static}) и (x_{fix} ; y_{max}^{static}).

$$\begin{cases} scale = \dfrac{y_{max}^{static} - y_{min}^{static}}{y_{max}^{template} - y_{min}^{template}} \\ displacement = y_{min}^{static} - y_{min}^{template} * scale \end{cases}$$

6. Из полученных ранее величин легко найти новые координаты точек шаблонного контура на текущем новом контуре:

$$\begin{cases} \mathrm{x}_{new=} x_{fix} \\ \mathrm{y}_{new=} y_{old} * scale + displacement \\ \mathrm{z}_{new=} z_{old} * scale \end{cases}$$

Строим подобным образом новые контуры с некоторым шагом по оси Ox и в итоге получим облако точек камня.

1.3 Построение полигональной модели камня

Таким образом мы получим облако точек (Рис.5), по которому нужно восстановить полигональную модель камня. Существует несколько способов сделать это:

1. Первый способ – наивный. В алгоритме построения облака точек мы строили новые контуры проходя по оси Ox с некоторым шагом. Если мы будем запоминать точки каждого такого контура, то на этапе построения полигональной модели можем просто соединить соседние точки каждого нового контура, а также соответствующие точки соседних контуров, образуя таким образом четырехугольные полигоны. Торцами такого полигонального камня будут являться два многоугольника ограниченных двумя крайними новыми контурами. Этот способ прост в исполнении, но имеет недостатки в виде неестественного представления модели камня, так как при недостаточно частом разбиении оси Ox и малом числе точек в шаблонном контуре части полигональной модели будут представлять собой прямой сдвиг разреза камня. При большом же числе точек число полигонов быстро растет, что при большом числе камней будет требовать большого числа вычислений для отображения всей стены.
2. Другой способ – использование алгоритмов основанных на

восстановлении нормалей к точкам облака и восстановлении поверхности Пуассона [6]. Однако такие алгоритмы требуют большого числа точек в облаке и далеко не всегда могут восстановить поверхность. Но результат работы (Рис.6) этих алгоритмов гораздо качественнее предыдущего варианта, так как полигоны сгенерированной модели не завязаны на отдельные контуры, которые мы получили на этапе построения облака точек.

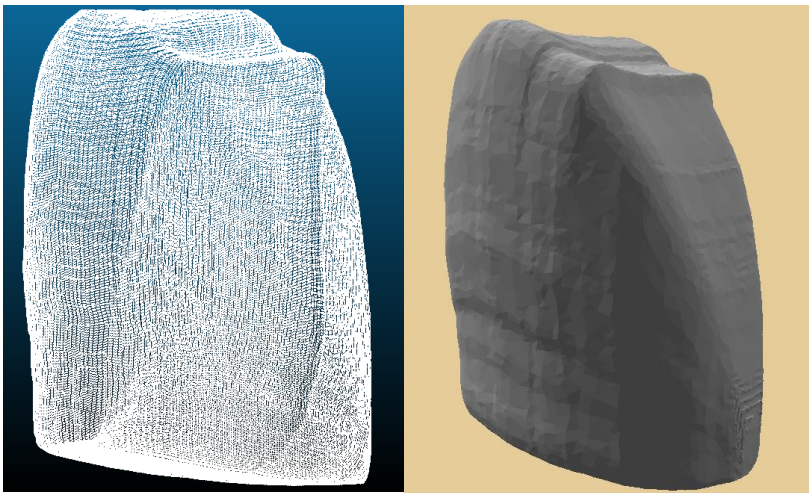


Рис.5 Рис.6 – облако точек и полигональная модель камня

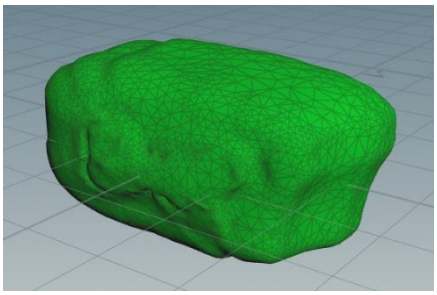
Теперь, когда мы получили все модели камней и их координаты мы можем расположить их на своих местах и получить объемную модель стены, где каждый камень является отдельным объектом.

Глава 2. Восстановление модели стены по малой информации

Поставим следующую задачу: пусть у нас имеются трехмерные модели камней (если у нас есть абрисы камней, то по алгоритму из предыдущей главы сможем получить 3D модели этих камней). Необходимо сгенерировать трехмерную модель стены по существующим моделям составляющих. При этом задаются следующие параметры стены:

- “Кривая”, по которой будет строиться стена – в виде упорядоченного набора точек в пространстве, которые последовательно соединяются между собой, образуя ломаную
- Фиксированная размер (длина, высота и ширина ограничивающего параллелепипеда) объекта, которым будет замощаться стена
- Высота стены

Тогда алгоритм восстановления стены по отдельным малым составляющим (Рис.7, Рис.8) будет выглядеть следующим образом:



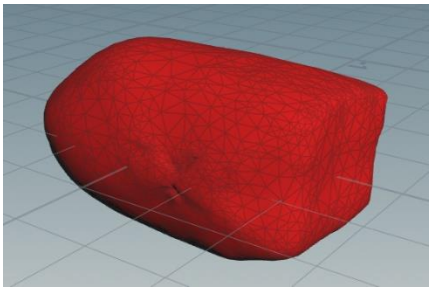


Рис.7, Рис.8 – модели камней

1. Спроецируем “кривую” на поверхность Oxy (учитывая, что “кривая” в нашем случае представлена, как последовательность точек, то это делается простым отбрасыванием z составляющей координаты узла ломаной)
2. Теперь зная размер элемента, которым будет замощаться стена, мы можем отложить на проекции ломаную из отрезков длины равной длине камня (r). Будем откладывать эти отрезки следующим образом:

a. Фиксируем начало звена ломаной в начале “кривой”

$$x_{current} = (x_0 ; y_0)$$

b. Последовательно перебираем отрезки $((x_i ; y_i) ; (x_{i+1} ; y_{i+1}))$ и ищем первый, с которым пересечется окружность радиуса r с центром в $x_{current}$.

c. Найдя такую точку, сохраняем получившееся звено ломаной $(x_{current} ; (x_{cross} ; y_{cross}))$ и переносим $x_{current}$ в точку пересечения

$$x_{current} := (x_{cross} ; y_{cross})$$

d. Повторяем операции a – c с новым положением $x_{current}$ и оставшимися звеньями “кривой”, не забывая при этом, что в одном звене r может уместиться несколько раз.

3. Таким образом, мы аппроксимировали “кривую” ломаной со звеньями одинаковой длины. На центры отрезков располагаются модели камней, отмасштабированные по заданному размеру, причем они ориентируются по направлению звена. В итоге получается первый ряд стены.
4. Следующий ряд строим на основе ломаной, построенной на шаге 2. Строим новую ломаную, концы звеньев которой лежат на центрах звеньев предыдущей ломаной.
5. Модели камней на втором ряду располагаем по центрам звеньев новой ломанной, ориентируя их по направлению звена. Таким образом получается второй ряд стены.
6. Последующие ряды строим, на основе ломаных первого и второго ряда, пока не достигнем высоты стены

Таким образом, мы получили стену, построенную на кривой на плоскости (Рис.9). Чтобы построить стену на кривой в пространстве (Рис.10), нужно просто от стены по кривой на плоскости откинуть составляющие, лежащие ниже исходной “кривой”, и составляющие, лежащие выше “кривой”, сдвинутой на высоту стены по оси Oz.

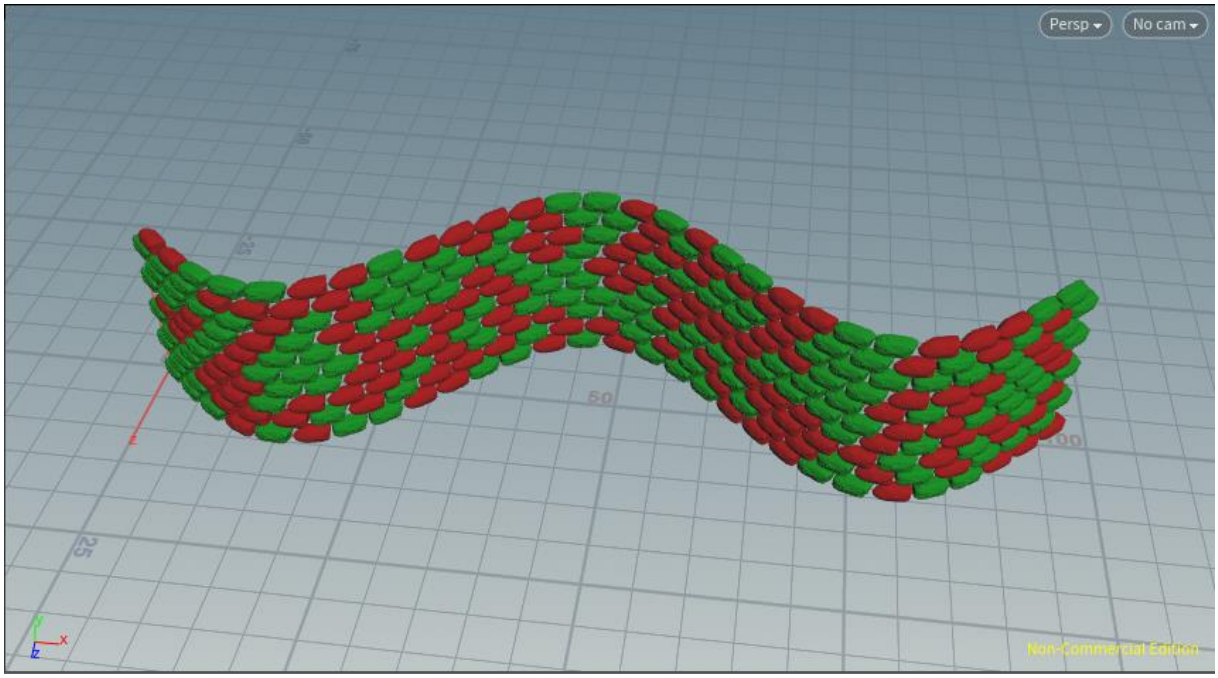


Рис.9 – модель стены по кривой на плоскости

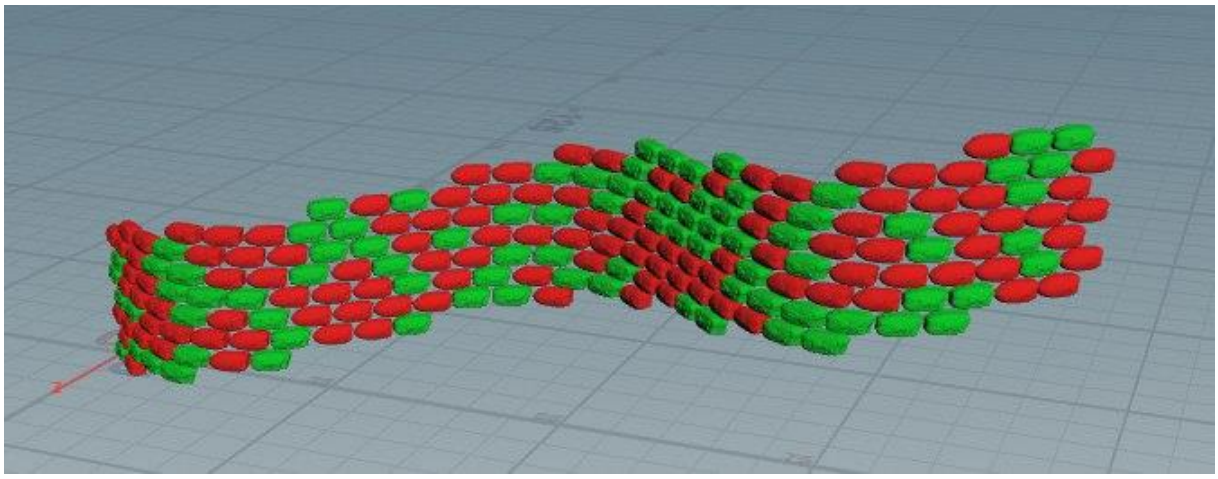


Рис.10 – модель стены по кривой в пространстве

Однако этот алгоритм имеет ряд недостатков. Здесь отсутствует возможность построения стены по замкнутой кривой. При попытке использовать в данном алгоритме замкнутую кривую возникнет ситуация с наложением

объектов друг на друга. На Рис.11 видно, как объекты зеленого и красного цвета смешиваются между собой.

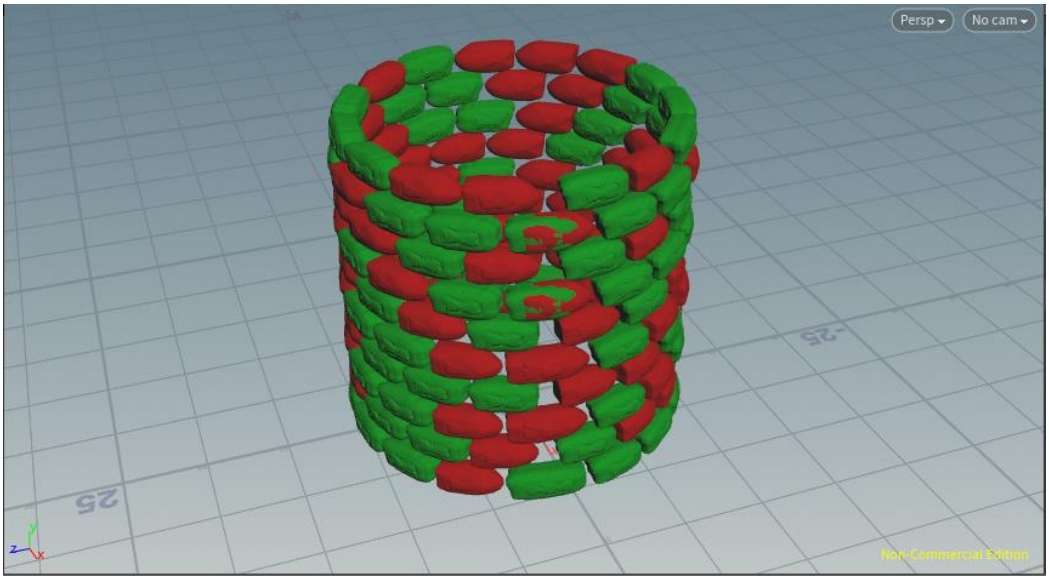


Рис.11 – модель стены по замкнутой кривой

К тому же не учитывается фактический размер составляющих (камни масштабируются под один размер) и отсутствует обработка кривых, проекция которых - есть самопересекающаяся кривая.

Глава 3. Восстановление модели стены с помощью генерации двумерного абриса

Поставим следующую задачу: необходимо на основе существующего двумерного абриса сгенерировать новый абрис, который будет подобен исходному, то есть будет учитывать некоторые его качественные характеристики. По такому новому двумерному изображению стены с помощью алгоритма, описанного в главе 1, можно будет восстановить трехмерную модель стены.

3.1 Сверточная нейронная сеть

Будем решать эту задачу с помощью глубинного обучения сверточных нейронных сетей [7, 8]. Сверточные нейронные сети – специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание образов.

Сверточная нейронная сеть (СНС) представляет собой последовательно соединенные слои свертки, активации и субдискретизации. В качестве входа СНС принимает несколько матриц изображения (обычно три матрицы, по матрице на каждый цветовой канал), а на выходе СНС производит вектор признаков.

Слой свертки – основной компонент СНС. Слой свертки включает в себя для каждого канала свой фильтр (ядро свертки), которым проходятся по всему входному слою и применяют операцию свертки (сумма попарных произведений соответствующих элементов на фильтре и входном слое). Весовые коэффициенты фильтра изначально неизвестны и устанавливаются в ходе обучения.

Смысл сверточного слоя заключается в уменьшении количества параметров, которые необходимо вычислить во время обучения. Так если, например, исходный слой имеет размеры 100x100 по 3 каналам (то есть 30000 нейронов на входе), а сверточный слой использует фильтры размером 3x3 и

переводит изображение в 6 каналов, то в процессе обучения нужно будет определить 9 весов ядра по всем сочетаниям входных и выходных каналов (то есть 9x3x6 = 162 параметров). И это гораздо меньше числа параметров полносвязной нейронной сети.

Слой активации Результат каждой свертки передается в качестве аргумента в нелинейную функцию активации. Слой активации обычно объединяют вместе со слоем свертки. Функция активации обычно определяется исследователем: обычно использовались сигмоиды:

$$f(x) = \frac{1}{(1 + e^{-x})}$$

или функции гиперболического тангенса:

$$\mathrm{f}(x) = \tanh(x) \text{ или } f(x) = |\tanh(x)|$$

Но сейчас большую популярность получила функция ReLU:

$$f(x) = \max(0, x)$$

позволяющая упростить вычисления и ускорить процесс обучения.

Слой субдискретизации (пулинг) – слой уплотнения карты признаков. Уплотнение происходит за счет нелинейного преобразования группы пикселей (обычно размера 2x2) в один пиксель (Рис.12). Обычно используется функция максимума (maxpool).

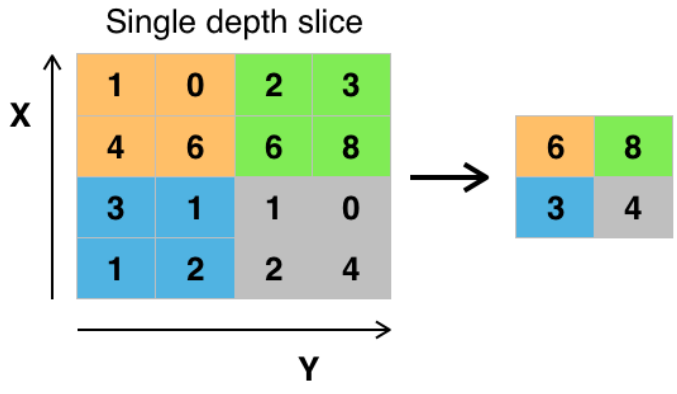


Рис.12 – пулинг с функцией максимума и фильтром 2x2

Пулинг позволяет уменьшить число исследуемых признаков, необходимость чего можно объяснить тем, что на предыдущей операции свертки были выявлены некоторые признаки и в дальнейшей обработке настолько

подробная информация уже не потребуется.

Самым популярным и простым способом обучения СНС является метод обратного распространения ошибки и его вариации.

3.2 Модификация сверточной нейронной сети

Обычная СНС, продвигаясь по иерархии слоев, преобразует изображение в представления, которые больше характеризуют фактическое содержание изображения (content), а не значения отдельных пикселей. Чтобы сгенерировать похожий двумерный абрис, нужно модифицировать СНС таким образом, чтобы выявлять не только стандартное содержания (content) изображения, но и так называемый стиль (style), характеризующий текстуру, которой покрывается фактическое содержимое изображения [9, 10].

Мы можем визуализировать информацию, которую содержит каждый слой об изображении, просто восстанавливая изображение по выходам фильтров (feature map) каждого слоя. Верхние слои сети содержат информацию высокого уровня с точки зрения объектов и их расположения во входном изображении, но не ограничивают точные значения каждого пикселя исходного изображения. Напротив, нижние слои просто воспроизводят точные значения пикселей исходного изображения. Поэтому параметры функций в верхних слоях мы считаем представлением содержимого (контента).

Чтобы получить представление о стиле входного изображения, мы построим отдельное пространство параметров, которое будет выявлять информацию о текстуре изображения. Это пространство будем строить поверх выходов фильтров каждого сверточного слоя (Рис.13). Оно будет содержать информацию о корреляции между выходами разных фильтров. Вычисляя корреляции параметров у нескольких слоев, мы получаем многослойное представления входного изображения, характеризующее его текстурную составляющую, а не контентную.

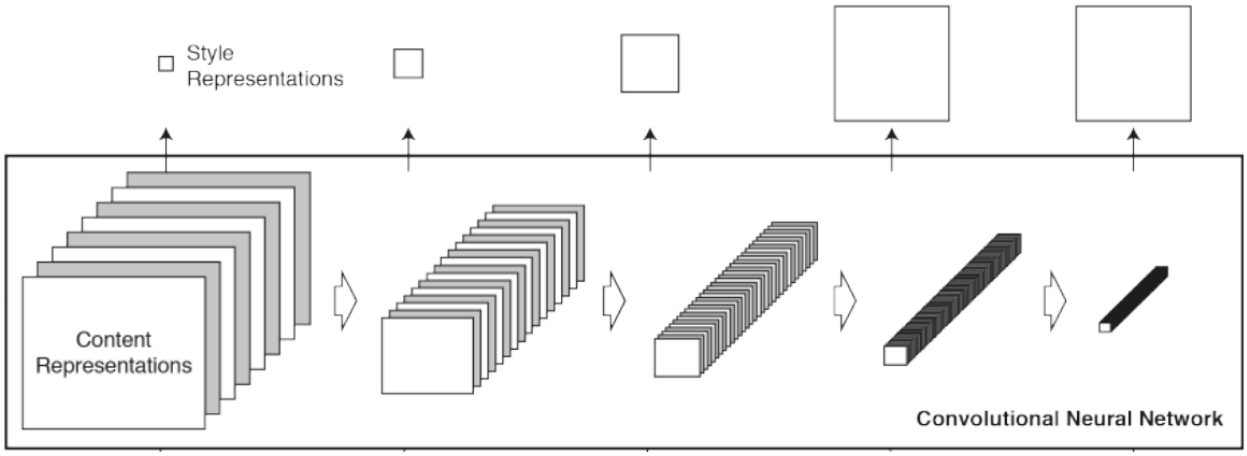


Рис.13 – модифицированная сверточная нейронная сеть

Таким образом, мы получаем сеть, в которой мы одновременно вычисляем характеристики контента и стиля изображения, по которым в дальнейшем с помощью метода обратного распространения ошибки можно будет генерировать абрисы, похожие на исходный. Опишем более подробно этот процесс.

В качестве основы модифицированной СНС возьмем уже обученную нейронную сеть VGG19 – СНС, выдающую большую точность распознавания объектов на изображении (Рис.14).

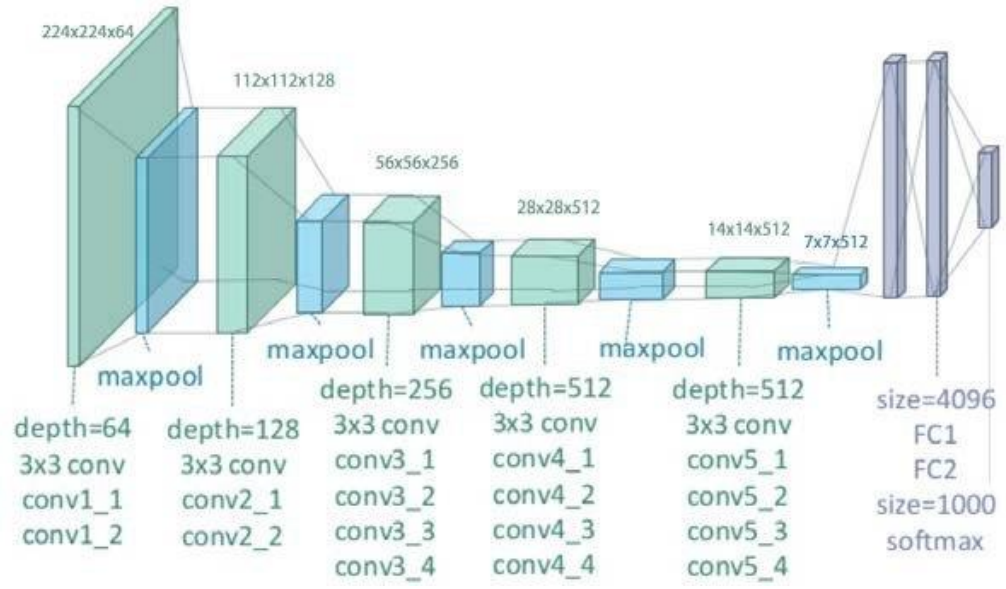


Рис14 – архитектура СНС VGG19

3.3 Генерация изображения с помощью модифицированной СНС

Каждый слой СНС представляет собой набор нелинейных фильтров, сложность которых возрастает с ростом уровня слоя сети. Следовательно входное изображение $\vec{x}$ закодировано в каждом слое результатом применения фильтров к этому изображению. У слоя с N_l различными фильтрами имеется N_l выходов фильтра, каждый размером M_l, где M_l – это произведение высоты выхода фильтра на его ширину. Тогда результат применения слоя l можно сохранить в матрице $F^l \in R^{N_l \, x \, M_l}$, где F_{ij}^l – это результат применения функции активации к применению i-го фильтра к элементу слоя l на позиции j.

Пусть $\vec{p}$ и $\vec{x}$ исходное и сгенерированное изображения соответственно, а P^l и F^l – результат применения слоя l к этим изображениям. Зададим функцию квадратичной ошибки между двумя результатами:

$$L_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{ij} \left(F_{ij}^l - P_{ij}^l\right)^2$$

Производная этой функции по результатам применения слоя равна:

$$\frac{\partial L_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & \text{если } F_{ij}^l > 0 \\ 0 & \text{если } F_{ij}^l < 0 \end{cases}$$

Используя производную, можно вычислить градиент $L_{content}$ по изображению $\vec{x}$ с помощью обратного распространения ошибки. Таким образом, мы можем менять изначально произвольное изображение $\vec{x}$ до тех пор, пока слои СНС не будут давать такой же результат, как и с изображением $\vec{p}$. Так мы получим изображения похожие по контенту.

Поверх каждого слоя СНС мы построили представление стиля изображения, которое считается как корреляция между выходами разных фильтров. Тогда характеристики стиля можно представить в виде матрицы Грама $G^l \in R^{N_l \, x \, N_l}$, где G_{ij}^l – скалярное произведение между векторами i и j выхода фильтра слоя l

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

Чтобы сгенерировать текстуру, которая соответствует стилю исходного изображения, нужно с помощью градиентного спуска перестроить любое изображение (например, белый шум) так, чтобы уменьшить среднее квадратичное расстояние между элементами матриц Грама для исходного и изменяемого изображения.

Пусть $\vec{a}$ и $\vec{x}$ – исходное и сгенерированное изображение соответственно, а A^{l} и G^{l} – результат применения модифицированного слоя стиля l к этим изображениям. Вклад этого слоя в общую ошибку будет равен:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{ij} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

и общая ошибка тогда будет равна:

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

где w_l – вес, определяющий вклад каждого слоя в общую ошибку. Производная E_l по выходам фильтра на слое l может быть представлена следующим образом:

$$\frac{\partial E_l}{\partial F_{ij}^{l}} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^{l})^{T} (G^{l} - A^{l})_{ji} & \text{если } F_{ij}^{l} > 0 \\ 0 \text{ если } F_{ij}^{l} < 0 \end{cases}$$

Используя производную, можно вычислить градиент E_l по изображению $\vec{x}$ с помощью обратного распространения ошибки. Таким образом, мы можем менять изначально произвольное изображение $\vec{x}$ до тех пор, пока модифицированные слои СНС не будут давать такой же результат, как и с изображением $\vec{a}$. Так мы получим изображения похожие по стилю.

Остается только объединить оба процесса и получим изображение похожее по контенту на $\vec{p}$ и по стилю на $\vec{a}$. Для этого введем комбинированную функцию ошибки, с помощью градиента которой и обратного

распространения ошибки можно вычислить необходимые изменения $\vec{x}$:

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

В результате получим нейросеть, которая по входному изображению стиля (Рис.15) и входному изображению контента (Рис.16) может выдать похожее изображение (Рис.17). Таким образом можно генерировать абрис по уже существующему изображению стиля и по регулируемому изображению контента, с помощью которого можно задавать некоторые характеристики стены, например расположение окон и дверей. По сгенерированному абрису строим стену с помощью алгоритма из главы 1.

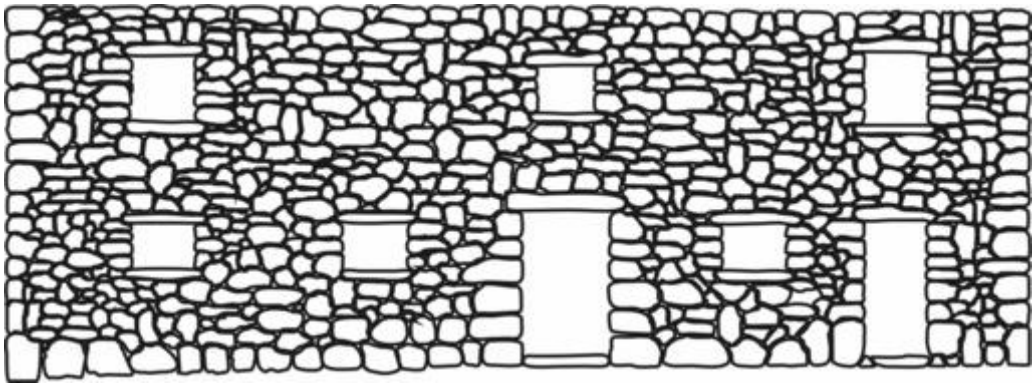


Рис.15 – изображение стиля

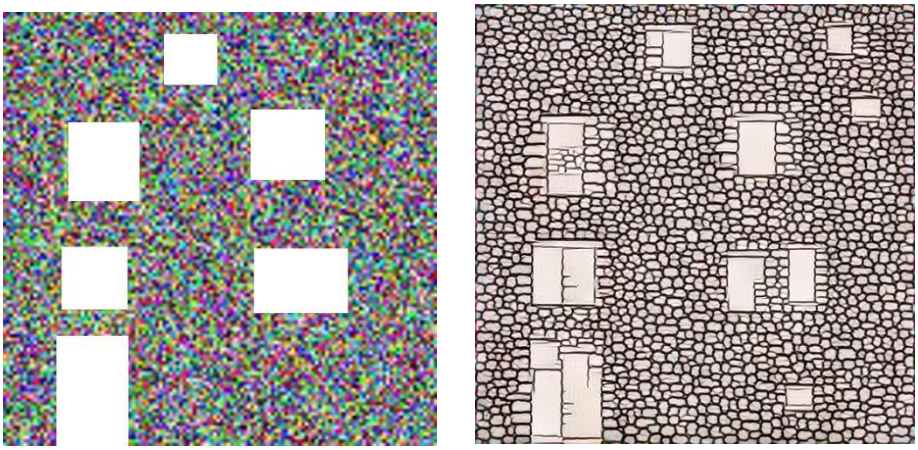


Рис.16 Рис.17 – изображение контента и сгенерированное изображение

Глава 4. Программная реализация

Все алгоритмы, описанные ранее, в основном были написаны на языке python. Алгоритм по преобразованию облака точек в полигональную модель был написан на языке C++, так как при его создании использовалась библиотека CGAL - библиотека алгоритмов вычислительной геометрии.

Для построения стены по кривой было решено использовать Houdini Engine [3]. Houdini Engine — это специальный программный пакет для работы с трехмерной графикой. Отличительной чертой Houdini Engine является то, что это среда визуального программирования, то есть моделирование в Houdini Engine может вестись с помощью процедурного программирования, используя так называемые ноды. Нода в Houdini Engine — это небольшая функция с несколькими входами и выходами, которая реализует какой-либо алгоритм (например, загрузка 3D модели из файла, наложение текстуры на модель, копирование объемного объекта и др.). Программирование с помощью нод представляет собой составление графа из нод, где выход одной ноды может служить входом другой. Такое программирование в Houdini Engine можно реализовывать с помощью графического редактора или используя язык программирования python.

Генерация двумерного абриса по уже существующему также производилась на языке python с помощью библиотеки Theano - библиотека численного вычисления в Python и Lasagne – библиотека для глубокого обучения. Вычисления проводились на основе архитектуры CUDA (Compute Unified Device Architecture) - программно-аппаратная архитектура параллельных вычислений от компании NVIDIA.

Выводы

В результате работы были разработаны несколько вариантов алгоритмов генерации трехмерной модели архитектурного памятника по его двумерному изображению. Каждый из алгоритмов был запрограммирован и протестирован. В итоге генерация двумерного абриса по тестовому изображению и дальнейшая реконструкция трехмерной модели стены с помощью сторонних решений по конвертации облака точек в полигональную модель показали себя лучше всего, так как эти алгоритмы были основаны на прорывных в сегодняшнее время сверточных нейронных сетях.

Результат работы программы можно улучшить, если совместить этот алгоритм с алгоритмом генерации стены по кривой в пространстве. Но результаты, представленные в текущей работе, уже могут служить в качестве некоторого инструмента, который поможет исследователям в сохранении информации о большем числе исторических памятников.

Список литературы

1. Швембергер, С. В., Щербаков, П. П., Горончаровский, В. А.. 3DS MAX Художественное моделирование и специальные эффекты. БХВ-Петербург. (2006)
2. Szolt. Blender Art 6 - Blender для архитектуры и игр. (2006)
3. Robert Magee. Houdini Foundations for film, TV and gamedev. (2018)
4. Rizvic, S., Okanovic, V., Sadzak, A.: Visualization and multimedia presentation of cultural heritage. Institute of Electrical and Electronics Engineers Inc. (2015)
5. Serra J. Image Analysis, Mathematical Morphology. (1982)
6. Michael Kazhdan, Matthew Bolitho, Hugues Hoppe Poisson Surface Reconstruction (2006)
7. K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition (2014)
8. V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition (2014)
9. Chuan Li, Michael Wand, Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis (2016)
10. Leon A. Gatys, Alexander S. Ecker, Matthias Bethge A Neural Algorithm of Artistic Style (2015)