Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Рендеринг 3D модели лица для системы идентификации по снимку

Курсовая работа студента 344 группы

Самарина Алексея Владимировича

Научный руководитель к.ф.-м.н, доц. А.Т.Вахитов

Санкт-Петербург

2013

Оглавление

Введение

В настоящее время, в области криминалистики и в различных системах идентификации и безопасности, актуальна задача распознавания лица по снимку. Самыми значимыми факторами, влияющими на сложность данной задачи являются внешние условия, такие, как освещение, положение камеры относительно объекта, и т. п. Одним из подходов, позволяющий производить распознавание при указанных условиях, является фиттинг 3D модели. Данный метод описан в ряде статей и представляет собой описание пространства, где каждый объект является линейной комбинацией, что, в свою очередь обеспечивает однозначность представления.

Постановка задачи

Одним из самых узких мест подобных подходов в плане производительности является рендеринг. Задачей данной работы является изучение технологий и алгоритмов, связанных с идентификацией лиц и реализовать эффективный механизм рендеринга, позволяющий получать изображения лиц при различных условиях освещения и положения объекта в пространстве. Таким образом, наша задача состоит в следующем. Создать механизм, способный получать представление различных лиц и интерпретировать полученное изображение однозначно, то есть с последующей возможностью идентификации полученной модели. Далее, реализовать механизм рендеринга при различных условиях освещения, положения лица в пространстве, а также, с учётом характеристик камеры.

Обзор существующих решений

В настоящее время существует множество продуктов, позволяющих осуществлять рендеринг трёхмерных моделей в том или ином виде. Большинство из них являются коммерческими продуктами, такие как **RenderMan** от компании Pixar и **Maxwell Render** от Next Limit Technologies. Они являются довольно мощными, дорогими и тяжеловесными системами, использование которых для системы идентификации оказалось бы неприемлемым по понятным причинам. Существуют также общедоступные системы и опубликованные схемы.

MeshLab

MeshLab является кроссплатформенным программным обеспечением с открытым исходным кодом. Является довольно мощным и гибким механизмом, позволяющим просматривать, редактировать различные форматы трёхмерных моделей, обеспечивать рендеринг с выводом на монитор двухмерного изображения модели.

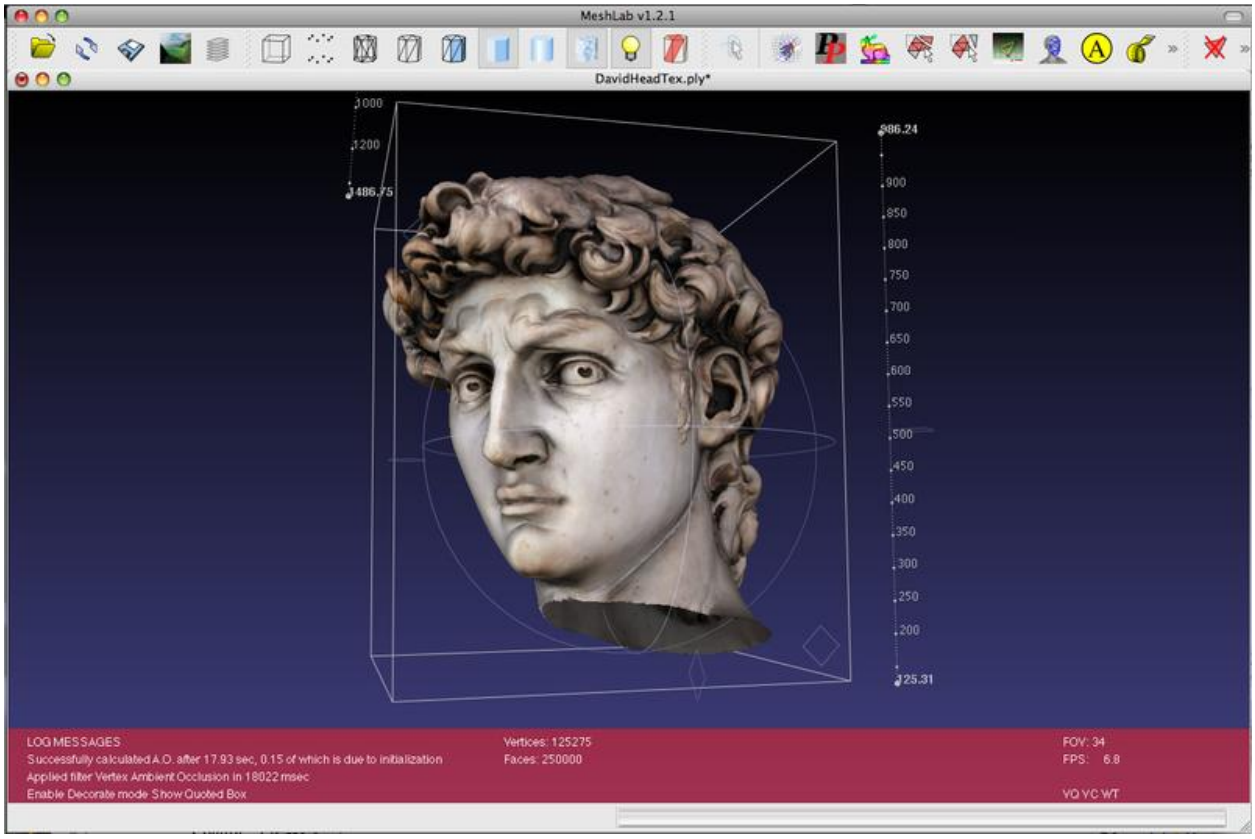

Рис. 1: Рендеринг 3D модели с помощью MeshLab.

Romdhani – Face Identification by fitting a 3DMM.

В статье Face Identification by fitting 3D Morphable Model авторов Sami Romdhani, Volker Blanz, Thamas Vetter описан механизм фиттинга, в том числе и рендеринг. Проблема данного подхода для рендеринга заключается в трудоёмкости, плохой масштабируемости и производительности прямой реализации, без использования специальных технологий.

Реализация

3D Morphable Model

Для реализации механизма распознавания, нам понадобится описать базу лиц, в которой каждое лицо будет представлено однозначно. Для этого возможно использовать следующий подход: создать линейное пространство, в котором каждое лицо будет разложено по некоторому базису. При этом лучше сразу создать два отдельных пространства – для формы и цвета соответственно. Теперь нам требуется выделить базисы для этих пространств. Для этого можно использовать следующий подход. Мы составим матрицу из всех векторов формы и текстуры, найдём их ковариационные матрицы. Далее, получив собственные числа и собственные вектора мы найдём направления и величины максимального разброса параметров, их мы и возьмём в качестве базисов нашего пространства. Рассмотрим подробнее вышеописанную схему. Для каждого лица, форма представляется как вектор из троек – координат точек лиц в пространстве, таким образом, для каждого лица мы имеем вектор формы - $s = (X_1; Y_1; Z_1; X_2; : : : ; Y_N; Z_N)$ и вектор текстуры – состоящий из трёх компонент для каждой точки соответственно – интенсивностей красного, зелёного и синего цвета соответственно - $t = (R_1; G_1; B_1; R_2; : : : ; G_N; B_N)$.

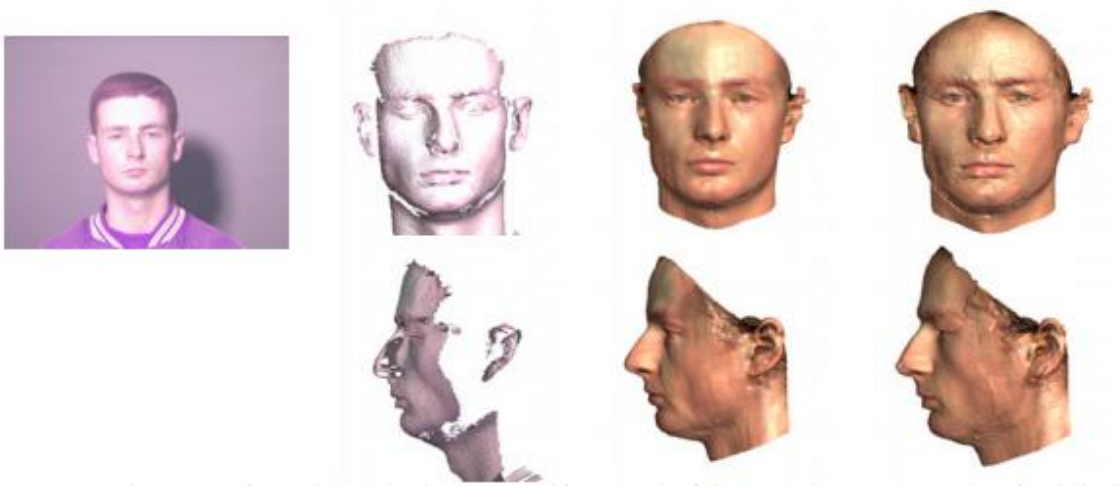

Рис. 2: Представление каждого лица как 3D модель из облаков точек, каждая из которых имеет координаты и трёх компонент цвета .

Когда мы получили представление каждого лица, как векторов текстуры и формы, для реализации линейного пространства, нам необходимо получить представление каждого из этих векторов в качестве некоторой линейной комбинации :

$$\mathbf{s} = \sum_{i=1}^{M} a_i \mathbf{s}_i^{ex}, \qquad \mathbf{t} = \sum_{i=1}^{M} b_i \mathbf{t}_i^{ex}$$

Для реализации этой идеи, нам требуется выбрать базис пространства. Чтобы сделать это наиболее эффективно, можно воспользоваться анализом главных компонент, то есть следует выбрать такие вектора, вдоль которых различия между лицами в нашей базе будет максимальным, выбрав их в качестве базиса нашего пространства, мы сможем добиться максимальной эффективности при минимальной размерности пространства, что повысит наши шансы на корректность работы нашего механизма распознавания. Для реализации данного подхода, мы составим ковариационную матрицу из всех векторов нашей базы лиц, найдём её собственные векторы, и собственные числа. Вектора, соответствующие наибольшим собственным числам и будут компонентами максимального различия, выбрав количество, необходимое для описания множества лиц нашей базы с помощью линейных комбинаций, мы и получим базис нашего пространства. Рассмотрим подробнее реализацию данного подхода. Для начала вычислим среднее лицо.

$$\overline{\mathbf{s}} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{s}_i^{ex}$$

Далее, найдём направления максимального различия – собственные вектора матрицы ковариации C, соответствующие максимальным собственным числам.

$$\mathbf{a}_i = \mathbf{s}_i^{ex} - \overline{\mathbf{s}}, \qquad \mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_M)$$

$$\mathbf{C} = \frac{1}{M} \mathbf{A}\mathbf{A}^{\mathrm{T}}$$

Теперь мы можем представить каждый элемент нашего пространства как сумму среднего лица и линейной комбинации собственных векторов.

$$\mathbf{s} = \overline{\mathbf{s}} + \sum_{i=1}^{M-1} \alpha_i \, \sigma_{s,i} \, \mathbf{s}_i \qquad \mathbf{t} = \overline{\mathbf{t}} + \sum_{i=1}^{M-1} \beta_i \, \sigma_{t,i} \, \mathbf{t}_i,$$

Или в матричной форме:

$$\mathbf{s} = \overline{\mathbf{s}} + S\boldsymbol{\alpha}, \qquad \mathbf{t} = \overline{\mathbf{t}} + T\boldsymbol{\beta}$$

Рендеринг.

Для наиболее эффективной реализации рендеринга было принято использовать механизм OpenGL, Microsoft Visual Studio 2010, C++. OpenGL – интерфейс библиотек для работы с низкоуровневой графикой, который реализуется поставщиками драйверов для видеокарт, поэтому рендеринг производится наиболее эффективно при использовании этой технологии, по сравнению со стандартным перемножением множества матриц для получения координат, однако, следует отметить, что и производительность, в этом случае зависит от конкретной реализации.

Рис. 3: Пример рендеринга произвольного элемента из базы лиц с использованием OpenGL.

Стоит отметить, что сам интерфейс OpenGL не предоставляет необходимое количество возможностей для полноценной реализации рендеринга, для чего используются библиотеки более высокого уровня, такие как glut.

Таким образом, общее решение состоит из рендеринга сгенерированной модели лица с известными параметрами с помощью библиотеки glut на монитор с последующим выбором нужного нам участка из буфера кадра, сохранения его в указанные массивы, и экспортом результата в необходимые форматы.

Отметим, что помимо сохранения результата рендеринга в заданные форматы, полезно иметь такую функциональность, как сохранение трёхмерной модели текущего лица в некоторый формат.

Одним из наиболее распространённых форматов для представления трёхмерных моделей объектов, является формат PLY. Данный формат наиболее подходит для экспорта нашей модели, так как может использовать триангуляционную сеть для задания формы объекта. Триангуляционная сеть используется для задания формы и корректного задания параметров освещения и в нашей модели. Для большей ясности рассмотрим простой пример – задание треугольника в данном формате:

```
ply
format ascii 1.0
comment this is a simple file
obj_info any data, in one line of free form text
element vertex 3
property float x
property float y
property float z
element face 1
property list uchar int vertex_indices
end_header
-1 0 0
 0 1 0
 1 0 0
3 0 1 2
```

Здесь мы видим заголовок, и описание структур – точек и рёбер, из которых состоит наша фигура. В данном случае это треугольник. Собственно говоря, модель лица и состоит из множества таких треугольников – триангуляционной сети, которая служит для определения параметров освещения и задания формы нашей модели в пространстве.

Рис. 4: Треугольник, описанный ранее в формате PLY.

После того, как мы задали трёхмерную модель, сгенерировав её с помощью линейной комбинации базисных векторов, при необходимости экспортировав её в трёхмерный формат, мы можем перейти к основной задаче – рендерингу, то есть получению двумерного изображения, согласно заданным условиям освещения, параметрам камеры, положением объекта в пространстве.

Таким образом, следующей нашей задачей будет воссоздание необходимых условий. Рассмотрим возможности OpenGL, с помощью которых и был реализован рендеринг.

Сама прорисовка реализуется с помощью задания графических примитивов, которые представлены на рисунке 5.

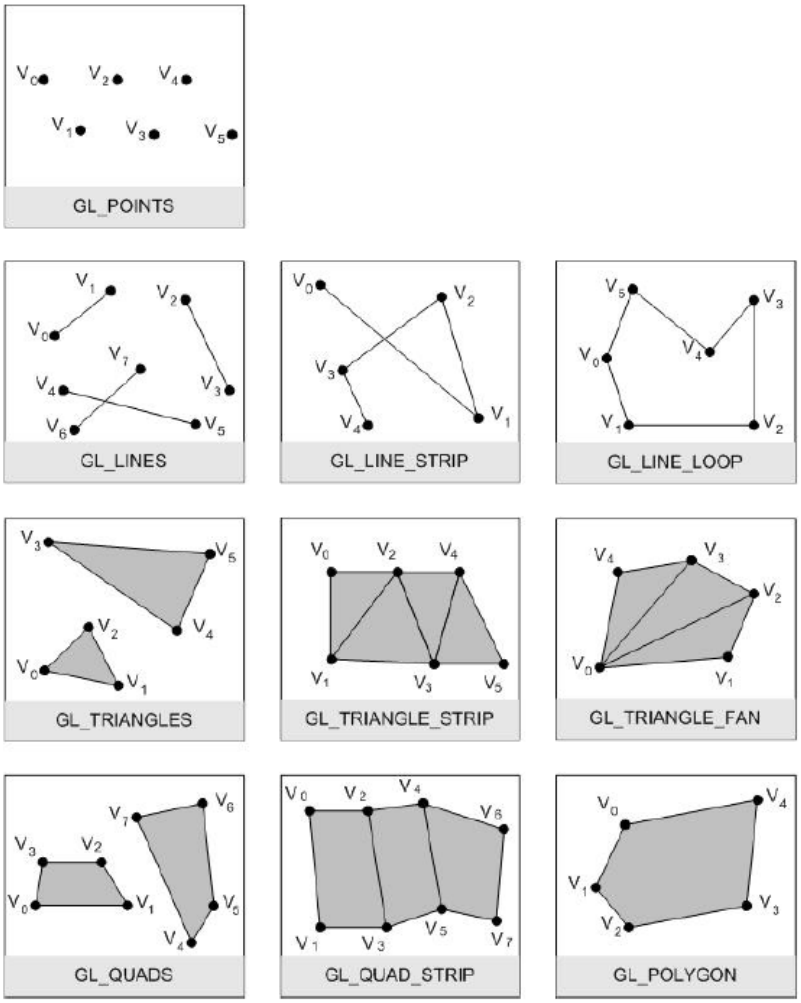

Рис. 5: Графические примитивы OpenGL.

Так же используются такие преобразования моделей как переносы и повороты. Заметим, что их можно трактовать, как движение камеры относительно объекта, так и перемещение модели в пространстве относительно камеры.

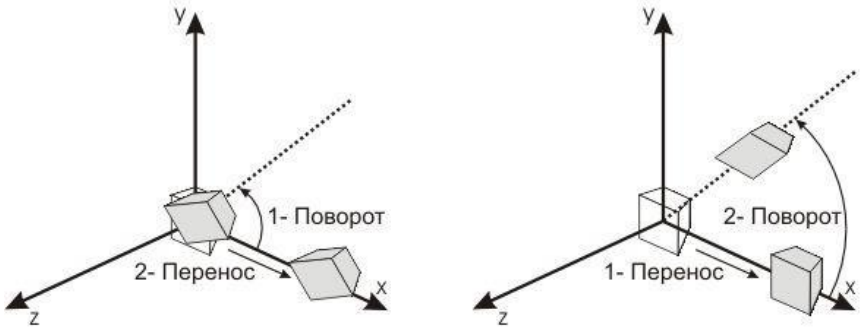

Рис. 6: Преобразования модели: перенос, поворот.

Так же необходимо поддерживать такие функциональности как масштабируемость, растяжение и сжатие модели.

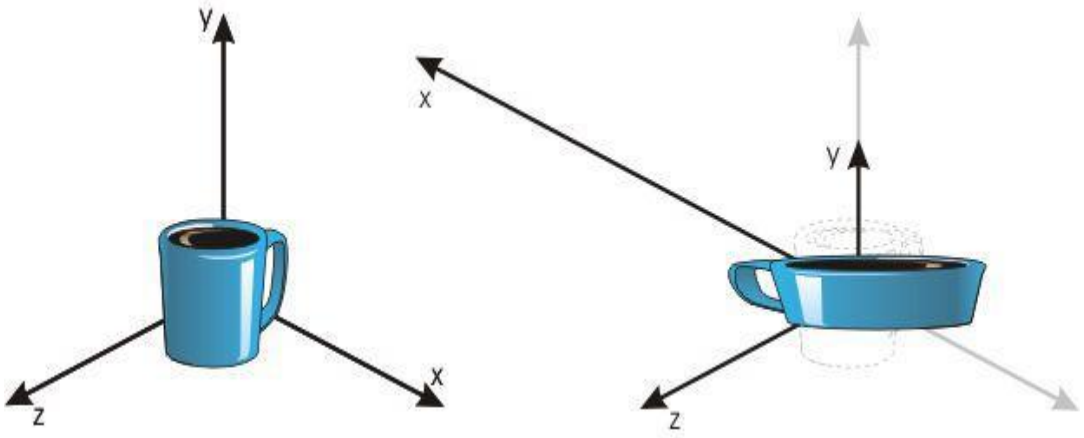

Рис. 7: Сжатие и растяжение модели.

Для реализации проецирования необходимо задать саму матрицу проектирования. В нашем методе мы пользуемся ортографической проекцией, то есть когда дальность от камеры не влияет на размер объектов в результирующем изображении.

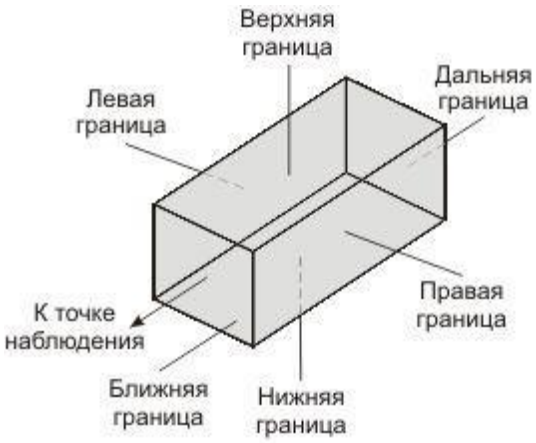

Рис. 8: Объём видимости ортографической проекции.

Что касается освещения, то созданная система предусматривает как фоновое, так и направленное освещение. Заметим, что для задания освещения, нам необходима триангуляционная сеть, с помощью которой, можно вычислить нормали, которые, в свою очередь необходимы для правильного расчёта влияния источников света на результирующее изображение.

Таким образом, задав все необходимые параметры, основные из которых были перечислены выше, мы получим результат рендеринга.

Рис. 9: Порядок обработки параметров при рендеринге.

Следует отметить, что OpenGL предоставляет программисту удобный интерфейс задания вышеперечисленных преобразований, который избавляет от необходимости пересчитывать и записывать матрицы руками.

Заметим, что полученный результат будет находиться в буфере кадра, откуда мы и вырежем нужный нам кусок, сохранив в последствии его в форматах, которые в дальнейшем могут быть распознаны библиотеками, с помощью которых осуществляется дальнейшие действия по идентификации.

Рис. 10: Пример результата рендеринга сгенерированного лица при заданных условиях освещения.

В итоге, предусмотрев все возможные условия получения изображения был реализован механизм рендеринга, основной идеей которого было пропустить сгенерированное изображение лица через конвейер OpenGL, и вырезать результат изображения из буфера кадра. Данный подход позволил использовать преимущества OpenGL, такие как удобство задания условий преобразования модели, высокая производительность.

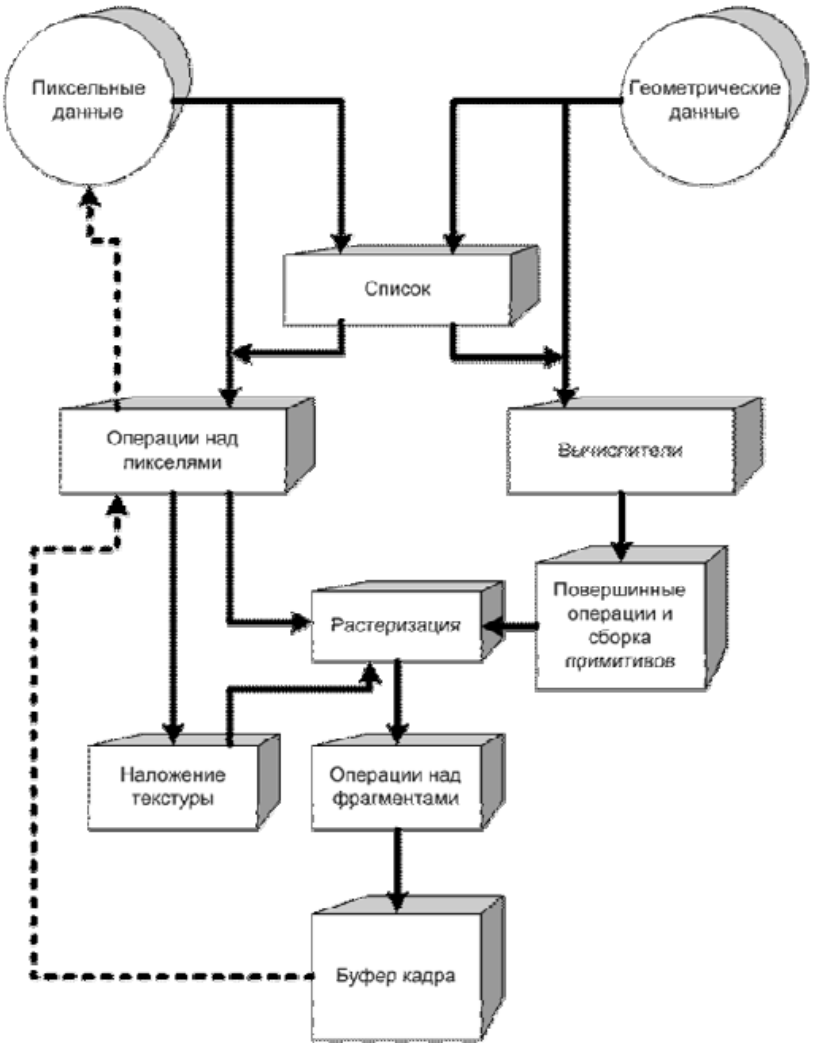

Рис. 11: Конвейер OpenGL.

Заключение

Результаты

В результате выполнения данной работы был прочитан ряд статей про алгоритмы компьютерной графики и методы распознавания и идентификации. Например, методы, основанные на выделении собственных векторов в качестве базиса, такие как Eigenfaces. Так же был изучен подход идентификации с помощью фиттинга трёхмерной модели, изучены основные подходы к решению задач рендеринга, изучено и проанализировано программное обеспечение, позволяющее решать данную задачу. Изучены возможности библиотеки алгоритмов компьютерного зрения OpenCV, изучен графический интерфейс OpenGL, с помощью которого и был реализован рендеринг трёхмерной модели лица.

Дальнейшее развитие

В дальнейшем возможно продолжить данную разработку в следующий направлениях:

- Оптимизация механизма рендеринга с целью расширения поддержки внешних условий.
- Использования данного механизма в различных системах идентификации и распознавания лиц.
- Расширение существующих интерфейсов и совместимости данного программного продукта.
- Дальнейшее развитие и оптимизация разработанной системы распознавания лиц.

Список литературы

1. B. Weyrauch, J. Huang,.. : Component-based Face Recognition with 3D Morphable Models.
2. Herve Abdi, Lynne J. Williams: Principal component analysis.
3. http://meshlab.sourceforge.net/
4. Matthew Turk, Alex Pentland: Eigenfaces for Recognition.
5. OpenGL Red Book.
6. Sami Romdhani, Volker Blanz: Efficient, Robust and Accurate Fitting of a 3D Morphable Model.
7. Sami Romdhani, Volker Blanz, and Thomas Vetter: Face identification by Fitting a 3D Morphable Model using Linear Shape and Texture Error Functions.
8. Theodore Papatheodorou: 3D Face Recognition Using Rigid and Non-Rigid Surface Registration.

Приложения

Приложение 1 – документация реализованных модулей.

data_manager – модуль ответственный за работу с данными

void load_mu (void) – производит загрузку данных о среднем (средней форме и текстуре)

void load_all_data (float *parameters) – загрузка данных о форме и текстуре лиц и генерация лица с параметрами, заданными в массиве parameters

void generate_face_with_parameters (float *parameters) – генерирует лицо с заданными в массиве parameters параметрами

void save_to_ply (void) – сохраняет текущую трёхмерную модель лица в формате ply

rendering_manager – данный модуль производит комплекс необходимых для реализации рендеринга операций

void set_light_diffuse_components (Glfloat r, Glfloat g, Glfloat b) – настройка параметров диффузного света: r – интенсивность красной компоненты, g – интенсивность зелёной компоненты, b – интенсивность синей компоненты

void set_light_direction_components (Glfloat x, Glfloat y, Glfloat z, Glfloat d) – установка параметров направленного света: x, y, z – компоненты вектора направления, d – индикатор бесконечной удалённости источника света

void set_scale_components (Glfloat x, Glfloat y, Glfloat z) – установка параметров сжатия-растяжения вдоль соответствующий осей: x, y, z – параметры сжатия-растяжения

void set_translation_components (Glfloat x, Glfloat y, Glfloat z) - установка параметров смещения модели: x, y, z – компоненты вектора смещения

void set_rotation _components (Glfloat c, Glfloat x, Glfloat y, Glfloat z) - установка параметров поворота модели: x, y, z – коэффициенты вращения относительно соответствующих осей, c – величина угла вращения

void init_lighting (void) - подключение освещения

void init_projection (void) – подключение всех параметров трансформации и проектирования модели

void import_to_tga (int w, int h) – импорт участка буфера кадра в формат tga: w, h – ширина и высота участка соответственно

void import_to_bmp (int w, int h) – импорт участка буфера кадра в формат bmp: w, h – ширина и высота участка соответственно
void render configured_mu_face(Glfloat c, Glfloat x, Glfloat y, Glfloat z, Glfloat w, Glfloat h) – рендеринг математического ожидания (среднего лица), повёрнутого на угол c относительно каждой оси координат с коэффициентами x, y, z, соответственно, w, h – размер области, в которой будет находиться результирующее изображение – ширина и высота, соответственно

void render configured_parametrised_face(Glfloat c, Glfloat x, Glfloat y, Glfloat z, Glfloat w, Glfloat h, float *parameters) – рендеринг модели лица, сгенерированной с коэффициентами из массива parameters, повёрнутой на угол c относительно каждой оси координат с коэффициентами x, y, z, соответственно, w, h – размер области, в которой будет находиться результирующее изображение – ширина и высота, соответственно

Страница проекта: https://code.google.com/p/simply-face-identification-system-based-on-fitting-3d-m-m/source/browse/trunk/proj/