\*Приветствие приёмной комиссии\*

\*Представление себя\*

\*Представление темы\*

Цель данной работы – реализовать программный инструмент для построения моделей трехмерных объектов, ориентированный на микроконтроллеры семейства STM32.

Задачи:

1. выбор и адаптация существующих алгоритмов трехмерной графики, позволяющих визуализировать трехмерную сцену, для выполнения на STM32;
2. реализация выбранных алгоритмов для создания трехмерной сцены;
3. исследование возможностей микроконтроллеров семейства STM32 для их применения при решении задач компьютерной графики.

Итак, почему для реализации 3d-движка были выбраны именно микроконтроллеры STM32?

Данная платформа нуждается в ПО, решающем задачи компьютерной графики. Микроконтроллеры STM32, оснащённые процессорами ARM, используются в отраслях, в которых нужна графика, начиная с промышленной автоматики и заканчивая пользовательской электроникой и устройствами интернета вещей. Дело в том, что микроконтроллеры хорошо выполняют роль интерфейсов в автоматизированных системах (развлечения, быт и т.д.). Этому способствует низкое энергопотребление, относительно низкая стоимость, гибкая и масштабируемая экосистема (периферия и среды разработки) и высокая производительность (меньше, чем у современных ПК, но достаточная для работы с графикой).

Стоит отметить, что STM32 имеет ряд аналогов, в том числе и от российских производителей, благодаря чему имеется возможность в будущем перенести разработки с STM32 на отечественную аппаратную платформу.

А какие есть существующие решения?

Для ответа на этот вопрос был проведён анализ существующих инструментов на GitHub.

Были библиотеки, решающие поставленную задачу (подробно см. в РПЗ) (Результаты их работы на слайде).

Однако все они обладают рядом существенных недостатков:

1. ориентированность исключительно на конкретные модели микроконтроллеров и дисплеев;
2. относительно низкая производительность за счёт отсутствия низкоуровневых оптимизаций;
3. способ задания исходных данных затрудняет использование графического инструмента;
4. отсутствует пользовательский интерфейс;
5. отсутствует масштабируемость.

Обладая этими недостатками, программный инструмент не может использоваться в полномасштабных проектах.

Отсюда следует необходимость разработки инструмента, не имеющего данных недостатков и качественно решающего поставленную задачу.

При выборе алгоритма удаления невидимых линий и поверхностей необходимо учитывать особенности решаемой задачи:

1. микроконтроллеры имеют относительно **небольшую вычислительную мощность**;
2. объём доступной памяти микроконтроллеров очень ограничен и составляет, как правило, **не более 0.5–2 Мб**.

Исходя из этих особенностей, можно сформулировать требования к алгоритмам:

1. **компактность** – алгоритмы должны задействовать минимальное количество памяти, используя структуры данных, не содержащие избыточной информации;
2. **простота и лаконичность** – алгоритмы должны быть простыми, чтобы итоговый исполняемый файл занимал как можно меньше места в памяти микроконтроллера;
3. **быстродействие** – микроконтроллер, не обладающий большими вычислительными мощностями, должен выполнять алгоритмы за приемлемое время.

Полный обзор алгоритмов с обоснованием, почему он подходит/не подходит для решения задачи. Сейчас я упомяну только 4 основных.

1. **Обратная трассировка лучей** – один из самых популярных алгоритмов. Но трассировка большого количества лучей является очень трудоёмким процессом – большие вычисления, которые микроконтроллер не потянет. Можно распараллелить алгоритм, но у микроконтроллера всего одно процессорное ядро. До свидания.
2. **Z-буфер** – тоже очень популярный. Вычисления проще, чем в трассировке лучей, но ему требуется больший объём памяти (порядка нескольких Мб). Для ПК это не проблема, а для микроконтроллера – явная причина отказаться от данного алгоритма. Можно использовать построчное сканирование, но тогда вырастет вычислительная сложность.
3. **Варнок** – менее популярный алгоритм. Его идея состоит в том, что на обработку областей изображения, содержащих мало информации, должно тратиться очень мало времени, и наоборот. Имеет меньшую вычислительную сложность, чем у Z-буфера, и не работает с большими буферами (глубины и кадра). В целом алгоритм подходит для решения задачи.
4. **Вейлер-Азертон** - производится попытка минимизировать количество шагов в алгоритме Варнока путем разбиения окна вдоль границ многоугольника. Но он основан на работе с двунаправленными циклическими списками и одноимённом алгоритме отсечения, поэтому по быстродействию он будет проигрывать Варноку.

Методы закрашивания:

1. простая закраска - один уровень интенсивности на грань;
2. закраска по Гуро - билинейная интерполяция интенсивностей;
3. закраска по Фонгу - билинейная интерполяция векторов нормалей.

Да, простая закраска применима при 3 условиях (…) и проигрывает Гуро и Фонгу в реалистичности, но Гуро и Фонг требуют большой объём вычислений, который микроконтроллер **не сможет выполнить за приемлемое время**.

Итак, для реализации был выбран алгоритм Варнока и простая закраска.

В РПЗ есть схема алгоритма и код реализации вместе с описанием деталей.

Основные шаги алгоритма:

Для каждого окна:

1. формирование массива полигонов;
2. выделение внешних и охватывающих полигонов;
3. удаление внешних полигонов;
4. разбиение окна на подокна, если
   1. не все многоугольники являются внешними и охватывающими;
   2. среди охватывающих многоугольников нет ближайшего к наблюдателю;
5. изображение содержимого окна.

**Диагра́мма развёртывания** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Deployment diagram*) в [UML](https://ru.wikipedia.org/wiki/UML) моделирует *физическое* развертывание артефактов на [узлах](https://ru.wikipedia.org/w/index.php?title=Node_(UML)&action=edit&redlink=1). Например, чтобы описать веб-сайт, диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, [JDBC](https://ru.wikipedia.org/wiki/JDBC), [REST](https://ru.wikipedia.org/wiki/REST), [RMI](https://ru.wikipedia.org/wiki/RMI)).

В центре событий – микроконтроллер. Ему нужен источник питания и программатор. Программатор загружает в микроконтроллер программу и модели для визуализации. В реализации данной схемы роль программатора будет исполнять ПК. В соответствии с нотацией, дисплей является частью микроконтроллера, а программа для микроконтроллера зависит от интерфейса дисплея, так как дисплеи у микроконтроллеров могут быть разными, т.е. иметь разный тип подключения к микроконтроллеру.

В качестве языка программирования был выбран язык C++:

1. **высокая скорость работы**; - лучше, чем у питона;
2. **строгая типизация**; - контроль качества кода без написания тестов;
3. **наличие библиотек для работы с микроконтроллерами семейства STM32 и периферийными устройствами**; - есть для многих языков, хоть для Haskell (платформа то популярная), но для C++ больше всего, так как это классический язык для программирования на микроконтроллерах;
4. **наличие большого количества материалов о разработке программного обеспечения для микроконтроллеров семейства STM32 на данном языке**; вытекает из п.3;
5. **возможность сертификации программ, написанных на данном языке**. – сертификация госов (ФСТЭК) важна для написания ПО в России и регистрации патентов, мб и для продвижения разработок;

В качестве среды разработки был выбран Visual Studio Code, обладающий большим количеством плагинов и инструментов для разработки программ на C++.

Есть код, работающий на ПК, а есть код, работающий на микроконтроллере, подключенном к ПК.

Первая часть ответственна за подготовку исходных данных для алгоритмов визуализации, вторая – за расчёты и вывод изображения на дисплей;

Если есть время, прокомментировать, что делает каждый модуль.

Для программы был разработан консольный интерфейс, а не графический. Такое решение было принято по причине выбора формата задания исходных данных, не требующего от пользователя большого количества действий. Также консольный интерфейс позволяет использовать разработанный программный инструмент другими программами. В будущем это позволит разрабатывать программное обеспечение для семейства микроконтроллеров STM32, использующее трёхмерную графику. Для запуска программы необходимо выполнить четыре шага: установить зависимости, подготовить исходные данные, подключить микроконтроллер к компьютеру и запустить рендеринг.

Для запуска рендеринга необходимо выполнить в командной оболочке команду вида

**sudo make view model=cube device=/dev/sdb target=/run/media/user/unit**

где **model** - название модели, **device** - файл подключенного устройства, **target** - точка монтирования устройства. Далее произойдёт компиляция программы, после чего она загрузится на микроконтроллер и выведет изображение на дисплей.

На слайде представлены как низкополигональные модели, так и высокополигональные.

Для программы, которая должна выполняться на микроконтроллере, обладающем сравнительно небольшой вычислительной мощностью, критически важен фактор быстродействия. Особенно он важен при изображении сцены, содержащей большое количество объектов. В связи с этим очень важно понимать, как оборудование, исполняющее разработанную программу, будет справляться с увеличением нагрузки и как при этом будет изменяться ресурсоёмкость реализованных алгоритмов.

Целью проводимых измерений будет проведение нагрузочного тестирования и установление зависимости между числом объектов на сцене и затратами времени и памяти на работу алгоритма визуализации.

Проводится попытка **выяснить зависимость** затрат времени и памяти от количества объектов (полигонов) на сцене и попытаться **определить предельную нагрузку** на микроконтроллер.

Было выбрано 3 модели.

Установление зависимостей производилось путём исследования процесса визуализации отдельных фрагментов изображения. Подробнее: в самом начале будет произведён замер времени визуализации небольшой части модели, а затем к ней постепенно добавляются новые полигоны до тех пор, пока число изображаемых полигонов не достигнет числа полигонов во всей модели и она не будет полностью изображена.

Результаты измерений показали, что зависимость затрат памяти на работу алгоритма от числа объектов на сцене является **линейной**.

А зависимость затрат времени, предположительно, является **кусочно-линейной**. Коэффициент пропорциональности между числом объектов на сцене и временем работы алгоритма не является одинаковым на различных участках графиков, так как различные случаи взаимного расположения полигонов обрабатываются с разной скоростью. Увеличение времени работы алгоритма при добавлении на сцену очередного полигона зависит от того, будет ли он пересекаться с полигонами, которые уже находятся на сцене.

Также проведённые измерения показали, что оборудование, исполняющее разработанную программу, равномерно справляется с большой нагрузкой, так как зависимость между числом объектов на сцене и ресурсоёмкостью алгоритма является линейной на всех участках графиков. По результатам тестов, максимальное количество объектов (полигонов), которое микроконтроллер семейства STM32 позволяет обработать, превышает 6000.

**Заключение**

Разработанная программа позволяет получать на экране дисплея изображение полигональной модели, заданной пользователем. При разработке были учтены недостатки существующих программных решений для аппаратной платформы STM32.

Графический инструмент, разработанный в рамках курсового проекта, имеет по меньшей мере два направления дальнейшего развития.

1. Разработка новых приложений, использующих разработанный графический инструмент. К этому располагают интерфейс программы и формат задания исходных данных.
2. Перенос разработок с STM32 на отечественные аппаратные платформы. Это возможно благодаря наличию аналогов семейства микроконтроллеров STM32 отечественного производства.

Спасибо за внимание

Можете задавать вопросы, я с удовольствием на них отвечу!