

---

# Renderer


---

Маркелов Александр  
Академический университет

---

# Постановка задачи

---

- Реализовать сru-linux-based-drm-kms рендер, использующий для прорисовки drm-gem и другие drm фрейм-буферы
- Возможности :
  - выбор нужного устройства и базовых настроек вручную ( например, настройка связок kms-crtc  kms-connectors)
  - поддержка управляемого пайплайна с расширяемыми стадиями обработки
- Зачем, есть же OpenGL?

Хочется понять, как графические библиотеки работают изнутри.

# Технические подробности (part 1)

---

Реализованный общий пайплайн ( описано с точки зрения пользователя библиотеки ) :

- Выбираем устройство обработки ( видеокарту )
- Получаем список connectors и crtc, формируем связи
- Для каждой связи создаём Context, передаём в него дополнительно число фрейм-буферов ( использовать/не использовать multiple buffering), а также указатель на объект реализованного класса, унаследованного от интерфейса Drawable
- Запускаем loop
- Теперь будут приходить коллбэки в метод onDraw(Context &), реализованный в нашем классе - тут прорисовываем очередной фрейм

# Технические подробности (part 2)

---

Реализованный пайплайн прорисовки модели :

- Base Stage ( загрузка модели , текстур и ресурсов с диска )
- Vertex Stage ( используется как транслятор координат модели в мировые , затем - в систему координат камеры, затем - в координаты отсечения; также используется как вершинный шейдер ; применяется к полигону)
- Rasterizator Stage ( на входе - полигон в координатах отсечения, на выходе - список экранных координат)
- Pixel Stage ( здесь определяется z координата и цвет пикселя; используется как фрагментный ( пиксельный) шейдер )
- Drawer Stage ( рисуется пиксель на экране )

# Новые знания

---

- Графический стек Linux
- Навыки низкоуровневой разработки
- Основы компьютерной графики
- Проективная геометрия
- Навыки работы с git

# Ссылки

---

<https://github.com/sanya1111/Renderer>