Topaz application example

Для корректного запуска приложения требуется поддержка нескольких дополнений :

GL_ARB_bindless_texture - дополнение обеспечивает механизм доступа к текстурным объектам в шейдерах без первоначального связывания каждой текстуры к ограниченному набору текстурных юнитов. Используя это дополнение, приложение может запрашивать 64-битный unsigned integer текстурный хендл для каждой текстуры, к которой необходимо получить доступ, и затем использовать этот хендл для прямого обращения в шейдерах. Дополнение значительно уменьшает количество внутренних GL - вызовов, необходимых для управления связыванием ресурсов. Полная документация доступна по ссылке https://www.opengl.org/registry/specs/ARB/bindless_texture.txt

GL_NV_command_list - дополнение добавляет несколько новых особенностей, разработанных чтобы обеспечить очень низкие накладные расходы на драйвер и использовать повторно GL - команды и изменение состояний. Из-за того, что *state objects* отражают состояние целого графического конвеера, ожидается что они могут быть пре-валидированы и эффективно выполнены. Полная документация доступна по ссылке http://developer.download.nvidia.com/opengl/specs/GL_NV_command_list.txt

GL_NV_uniform_buffer_unified_memory - дополнение обеспечивает механизм обращения к *uniform buffer* - ам, используя GPU адрес. Операция *glBindBuffer(GL_UNIFORM_BUFFER, buffer)* одна из наиболее частых и дорогих операций во многих GL - приложениях из-за кеширования указателей. Цель данного дополнения сделать доступным для приложения определять uniform buffer state, который облегчит оверхед связывания объектов и управление памятью в драйвере. Полная документация доступна по ссылке https://www.opengl.org/registry/specs/NV/uniform_buffer_unified_memory.txt

GL_NV_shadow_samplers_cube - дополнение добавляет возможность делать выборку из CUBE_MAP текстуры в шейдере. Полная документация доступна по ссылке https://www.khronos.org/registry/gles/extensions/NV/NV_shadow_samplers_cube.txt

GL_ARB_shading_language_include – дополнение добавляет директиву #include в шейдеры, чтобы позволить повторное использование одного и того же кода. Полная документация доступна по ссылке https://www.opengl.org/registry/specs/ARB/shading_language_include.txt

Описание функций

Название функции	Описание работы
initRendering	Проверка наличия необходимых дополнений; компиляция шейдеров; загрузка 3D объектов и текстур
initFramebuffers	Инициализация color буфера и depth буфера; инициализация frame буфера для отрисовки сцены
initScene	Триангуляция объектов сцены; инициализация и аллокация памяти на GPU под VBO, IBO, UBO для всех объектов сцены
initCommandList	Инициализация команд-токенов, отвечающих за отрисовку объектов сцены; набор токенизированных команд в один token буфер
updateCommandListState	Установка необходимых состояний графического конвеера; захват текущего набора состояний, используя функцию glStateCapture; создание, компиляция токенизированного списка команд
draw	Отрисовка объектов сцены двумя способами: glDrawElements и glCallCommandList

initCommandList

В initCommandList осуществляется инициализация токенизированных-команд, соотвествующих различным классам, отвечающих за отрисовку объектов сцены (NvTokenVBO, NvTokenIBO, NvTokenUBO, NvTokenLineWidth, NvTokenViewport, NvTokenScissor) и регистрация размеров токенов.

Для каждой токенизированной команды, которая в последствии будет сложена в один командный список, необходимо узнать название, за которую отвечает этот токен, с помощью функции glGetCommandHeaderNV.

Список названий доступных команд

TERMINATE_SEQUENCE_COMMAND_NV
NOP_COMMAND_NV
DRAW_ELEMENTS_COMMAND_NV
DRAW_ARRAYS_COMMAND_NV
DRAW_ELEMENTS_STRIP_COMMAND_NV
DRAW_ARRAYS_STRIP_COMMAND_NV
DRAW_ELEMENTS_INSTANCED_COMMAND_NV
DRAW_ARRAYS_INSTANCED_COMMAND_NV
ELEMENT_ADDRESS_COMMAND_NV
ATTRIBUTE_ADDRESS_COMMAND_NV

UNIFORM_ADDRESS_COMMAND_NV
BLEND_COLOR_COMMAND_NV
STENCIL_REF_COMMAND_NV
LINE_WIDTH_COMMAND_NV
POLYGON_OFFSET_COMMAND_NV
ALPHA_REF_COMMAND_NV
VIEWPORT_COMMAND_NV
SCISSOR_COMMAND_NV
FRONT_FACE_COMMAND_NV

В initCommandList создается набор состояний, с помощью функции glCreateStatesNV. Текущий набор состояний графического конвеера может быть «захвачен» для будущего использования с новым набором команд отрисовки.

glCreateCommandListNV создает скомпилированный токенизированный список. Скомпилированный список нужен для дальнейшей оптимизации и эффективного повторного использования.

Далее идет создание токенов (объектов классов, представляющих токены) и запись их в токенизированный список с текущим состоянием. Если состояние меняется, то ведется запись в список, вычисляются и запоминаются параметры: offset, size, framebuffer и текущий набор состояний.

- offset - сдвиг относительно начала токенизированного списка, при смене состояния графического конвеера offset становится равным размеру токенизированного списка.
- size – разница между текущим размером токенизированного списка и параметром offset
- framebuffer - текущий id frame буфера, в который рендерится сцена
- текущий набор состояний будет «захвачен» командой glStateCapture

Выделяется память на GPU под токенизированный список команд tokenData с помощью команды glNamedBufferStorageEXT.

Для токенизированного списка, который будет скомпилирован в бинарное представление, необходимо к offset добавить адрес начала массива tokenData, куда был записан токенизированный список.

updateCommandListState

Производим установку необходимых состояний графического конвеера, захватываем текущий набор состояний, используя функцию glStateCapture.

Необходимо создать, скомпилировать и сделать доступным для использования токенизированный список команд.

Используем следующий набор GL команд:

- Функция glCommandListSegmentsNV указывает, что command list будет иметь определенное количество (равное segments) упорядоченных последовательностей команд, которые list будет ставить в очередь.
- Функция glListDrawCommandsStatesClientNV устанавливает текущие параметры offsets, sizes, states, fbos.
- Функция glCompileCommandListNV компилирует токинизированный список в бинарное представление для будущего вызова отрисовки.

Функцию updateCommandListState нужно вызывать каждый раз, когда текущее состояние графического конвеера отличается от захваченного (captured) состояния.

draw

Обновляем UBO с помощью команды glNamedBufferSubDataEXT для всех объектов сцены. Вызываем команду отрисовки для текущего токенизированного списка команд glCallCommandListNV.

Weight Blended OIT

Приближенный алгоритм для смешивания прозрачных объектов сцены, без предварительной сортировки объектов по глубине. Из-за того, что не производится сортировка объектов по глубине, алгоритм работает быстрее, по сравнению с аналогами. Weight Blended OIT является приближенным алгоритмом, поэтому дает погрешности при смешивании полупрозрачных объектов сцены. Аналогичный алгоритм Depth Peeling дает меньшую погрешность, но большее время работы.

Алгоритм Weight Blended OIT состоит из двух основных этапов:

- geometry pass
- composition pass

Перед тем, как пройти два этапа алгоритма Weight Blended OIT необходимо отрендерить в текстуру непрозрачные объекты сцены.

«Geometry pass» заключается в рендеринге прозрачных объектов сцены сразу в две текстуры. Первая текстура будет содержать информацию о RGBA компонентах сцены, вторая текстура будет содержать информацию только об альфа канале сцены.

В первой текстуре в каждой компоненте будет храниться сумма вида

$$\sum_i vec4(srcRGB_i * srcAlpha_i, srcAlpha_i)$$

Во второй текстуре в каждой компоненте будет храниться произведение вида

$$\prod_i vec4(1 - srcAlpha_i)$$

«Composition pass» заключается в смешивании непрозрачных объектов сцены с прозрачными. Информация о прозрачных объектах сцены была получена на предыдущем шаге «Geometry pass».

Замешивание происходит по формуле

$$finalColor_{rgb} = srcAlpha * transparentRGB + (1 - srcAlpha) * opaqueRGB$$