# Компьютерная графика

Курс лекций

Тема №10. Оптимизация приложений OpenGL.

# Организация приложения

- высокоуровневая оптимизация: необходим поиск компромисса между качеством и производительностью, например:
  - отображение геометрии сцены с низким качеством (с уменьшенным количеством примитивов) во время анимации, и с наилучшим качеством (полностью) в моменты остановок;
  - представление объектов, располагающихся далеко от наблюдателя, моделями пониженной сложности
  - объекты, которые находятся полностью вне поля видимости, могут быть эффективно отсечены без передачи на конвейер OpenGL с помощью проверки попадания ограничивающих их простых объемов (сфер или кубов) в объём видимости;
  - во время анимации можно отключить псевдотонирование (dithering), плавную заливку, текстуры;
- низкоуровневая оптимизация: использование структур данных, которые могут быть быстро и эффективно переданы на конвейер OpenGL.

# Оптимизация вызовов OpenGL

- стратегии оптимизации:
  - передача данных в OpenGL;
  - управление обработкой вершин в графическом конвейере;
  - управление растеризацией;
  - управление текстурированием;
  - управление очисткой буферов;
  - минимизация числа изменений состояния OpenGL.

# Оптимизация вызовов OpenGL: передача данных

- использование связанных примитивов (**GL_LINES**, **GL_LINE_LOOP**, **GL_TRIANGLE_STRIP**, **GL_TRIANGLE_FAN** и **GL_QUAD_STRIP**);
- использование массивов вершин (**glVertexPointer, glNormalPointer, glColorPointer, glInterleavedArrays, glDrawArrays, glArrayElement**);
- задание необходимых массивов одной командой (**glVertexPointer / glNormalPointer / glColorPointer → glInterleavedArrays**);
- использование индексированных примитивов (**glDrawElements**);
- последовательное хранение в памяти данных о вершинах (влияет на скорость обмена между основной памятью и графической подсистемой);
- использование векторных версий **glVertex, glColor, glNormal и glTexCoord**;
- уменьшение сложности примитивов (при сохранении наилучшего соотношения качества и производительности, в частности, при использовании текстурирования);
- использование дисплейных списков (могут храниться в памяти графической подсистемы);
- исключения указания ненужных атрибутов вершин (например, **glNormal** при выключенном освещении);
- минимизация количества лишнего кода между **glBegin/glEnd**.

# Оптимизация вызовов OpenGL: управление обработкой вершин в графическом конвейере

- освещение:
  - исключить использование локальных источников света, т.е. координаты источника должны быть в форме (x,y,z,0);
  - исключить использование двухстороннего освещения (two-sided lighting);
  - исключить использование локальной модели освещения;
  - исключить частую смену параметра материала **GL_SHININESS**;
  - рассмотреть возможность заранее просчитать освещение (можно получить эффект освещения, задавая цвета вершин вместо нормалей);
- отключение нормализации векторов нормалей (glEnable / glDisable(GL_NORMALIZE));
- использование связанных примитивов (**GL_LINES**, **GL_LINE_LOOP**, **GL_TRIANGLE_STRIP**, **GL_TRIANGLE_FAN**, и **GL_QUAD_STRIP**).

# Оптимизация вызовов OpenGL: управление растеризацией

- отключение теста на глубину (фоновые объекты, например, могут быть нарисованы без теста на глубину, если они визуализируются первыми);
- отсечение (отбрасывание) обратных граней полигонов до растеризации (glEnable(GL_CULL_FACE));
- минимизация лишних операций с пикселями (маскирование, альфа-смешивание и другие попиксельные операции могут занимать существенное время на этапе растеризации);
- уменьшение размера окна или разрешения экрана (в случае, если меньшие размеры окна или меньшее разрешение экрана приемлемы).

# Оптимизация вызовов OpenGL: управление текстурированием

- использование эффективных форматов хранения изображений (**GL_UNSIGNED_BYTE**);
- объединение текстур в текстурные объекты или дисплейные списки (при использовании нескольких текстур позволяет графической подсистеме эффективно управлять размещением текстур в видеопамяти)
- исключение использования текстур большого размера;
- комбинирование небольших текстур в одну большего размера и изменение текстурных координат для работы с нужной подтекстурой (позволяет уменьшить число переключений текстур);
- при использовании анимированных текстур для обновления образа текстуры - вызов команд **glTexSubImage2D** или **glTexCopyTexSubImage2D** вместо **glTexImage2D**.

# Дисплейные списки (display lists)

- если несколько раз производится обращение к одной и той же группе команд, то их можно объединить в *дисплейный список*, и вызывать его при необходимости;
- дисплейные списки в оптимальном, скомпилированном виде хранятся в памяти сервера, что позволяет рисовать примитивы в такой форме максимально быстро;
- в то же время большие объемы данных занимают много памяти, что в свою очередь влечет падение производительности (большие объемы (больше нескольких десятков тысяч примитивов) лучше рисовать с помощью массивов вершин;
- работа с дисплейными списками:
  1. создание дисплейного списка: **glNewList() / glEndList()**.
     - идентификация списка: целое положительное число
     - режим обработки списка:
       - **GL_COMPILE:** команды записываются в список без выполнения
       - **GL_COMPILE_AND_EXECUTE:** команды сначала выполняются, а затем записываются в список
  2. вызов списка(ов): **glCallList() / glCallLists()**
  3. удаление дисплейного списка: **glDeleteList()**

# Массивы вершин

- задание массивов:
  - массив координат вершин:

    **glVertexPointer (GLint size, GLenum type, GLsizei stride, void* ptr)**
  - массив координат нормалей:

    **glNormalPointer (GLenum type, GLsizei stride, void* ptr)**
  - массив цветов:

    **glColorPointer (GLint size, GLenum type, GLsizei stride, void* ptr)**
  - массив текстурных координат:

    **glTexCoordPointer (GLint size, GLenum type, GLsizei stride, void* ptr)**
  - одновременно несколько массивов:

    **glInterleavedArrays(GLenum format, GLsizei stride, const GLvoid * pointer);**
- определение используемых массивов:

  **glEnableClientState(GLenum array) / glDisableClientState(GLenum array)**
- использование сформированных массивов для отрисовки:
  - отдельного элемента:

    **void glArrayElement(GLint i)**
  - набора примитивов по данным из массивов:

    **void glDrawArrays(GLenum mode, GLint first, GLsizei count);**
  - набора примитивов по данным из массивов (по индексам):

    **void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid * indices);**

# Лабораторная работа №7 – оптимизация приложений OpenGL

- цель работы: изучение эффективных приемов организации приложений и оптимизации вызовов OpenGL.
- задача: оптимизация приложения OpenGL, созданного в рамках предыдущей лабораторной работы, на основе выбора наиболее эффективных методик. (см. Баяковский Ю.М., Игнатенко А.В. Начальный курс OpenGL.- М.: «Планета Знаний», 2007.- 221с. – Глава 9).
- обязательно использовать дисплейные списки и массивы вершин.
- оценка применимости выбранного метода оптимизации приложения OpenGL должна осуществляться на основании измерения производительности (с помощью функции WinAPI QueryPerformanceCounter или подобных ей в других ОС).
- результаты замеров оформить в табличном виде.

# Вопросы к экзамену

- Оптимизация приложений OpenGL.