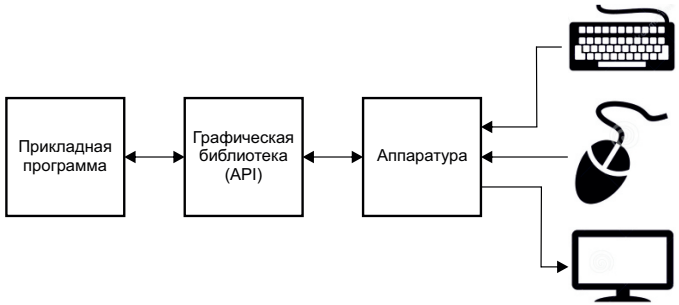
Модуль 1 «Основы компьютерной геометрии» Лекция 4 «Библиотеки графических функций. Введение в WebGL»

к.ф.-м.н., доц. каф. ФН-11, Захаров Андрей Алексеевич,
ауд.:930а(УЛК)
моб.: 8-910-461-70-04,
email: azaharov@bmstu.ru

МГТУ им. Н.Э. Баумана

29 сентября 2024 г.

Графические библиотеки



Графическая библиотека — это множество функций, которые образуют интерфейс (*abstraction layer*) между прикладной программой (или языком программирования, с помощью которого разрабатывается программа) и графической системой. Этот интерфейс называют программным интерфейсом приложения компьютерной графики (CG API — Computer-Graphics Application Programming Interface). Для программиста, занимающегося разработкой прикладной программы, существует только API, и он избавлен от необходимости вникать в подробности работы аппаратуры и программной реализации функций графической библиотеки. В схеме, приведённой на рисунке, по одну сторону от API располагается прикладная программа, а по другую — комбинация программных и аппаратных средств, которая реализует функции, специфицированные в API, и формирует изображение на экране.

Использование API позволяет разработчикам программ делать их универсальными, независимыми от низкоуровневых команд конкретного графического адаптера.

Стандарты разработки графических библиотек

Графические библиотеки должны находить баланс между слишком высоким и слишком низким уровнем абстракции. С одной стороны, библиотека должна скрыть различия между продуктами различных производителей (или между различными продуктами одного производителя) и системно-специфическими чертами, такими как разрешение экрана, архитектура процессора, установленная операционная система, и так далее. С другой стороны, уровень абстракции должен быть достаточно низким, чтобы программист мог получить доступ к базовому аппаратному обеспечению. Если графическая библиотека представляет слишком высокий уровень абстракции, то легко создавать универсальные программы, но очень трудно использовать расширенные возможности графического аппаратного обеспечения, которые в нее не были включены. Низкие уровни абстракции распространены, например, в игровых консолях, но они не применимы для графической библиотеки, которая должна поддерживать устройства, начиная от мобильных телефонов до игровых ПК и заканчивая высокопроизводительными профессиональными графическими рабочими станциями.

Стандарты разработки графических библиотек

Графические функции в любом пакете, как правило, задаются как набор описаний, которые не зависят от какого бы то ни было языка программирования. Затем задаётся привязка к языку программирования высокого уровня (C, C++, Java, Fortran, Delphi, Basic,. . .). Эта привязка задаёт синтаксис, позволяющий пользоваться различными графическими функциями этого языка. Каждая привязка к языку задаётся так, чтобы можно было максимально использовать соответствующие возможности языка: типы данных, синтаксис задания параметров и обработки ошибок.

Стандарты разработки графических библиотек

Рендеринг — это процесс создания изображения на основе модели с помощью компьютерной программы. Существуют различные виды рендеринга: программный и аппаратный рендеринг, удаленный и локальный рендеринг, а также рендеринг в режиме сохранения и немедленного режима.
Программный рендеринг использует для выполнения необходимых вычислений центральный процессор (ЦП) компьютера. Аппаратный рендеринг выполняет вычисления 3D-графики с помощью графических процессоров. С технической точки зрения аппаратный рендеринг намного эффективнее программного, поскольку первый предполагает использование специализированного оборудования для обработки необходимых операций. Программный рендеринг может быть более универсальным, т.к. не требует наличия аппаратных возможностей.
Процесс рендеринга может происходить локально или удалённо.
Когда изображение, которое необходимо визуализировать, слишком сложное, визуализацию лучше производить на удалённом сервере. Это касается 3D-анимационных фильмов, в которых серверы с большим количеством аппаратных ресурсов визуализируют сложные сцены.
Противоположный этому подход имеет место при локальном рендеринге. Вся обработка, необходимая для получения изображения, выполняется. локально с использованием графического оборудования клиента.

Стандарты разработки графических библиотек

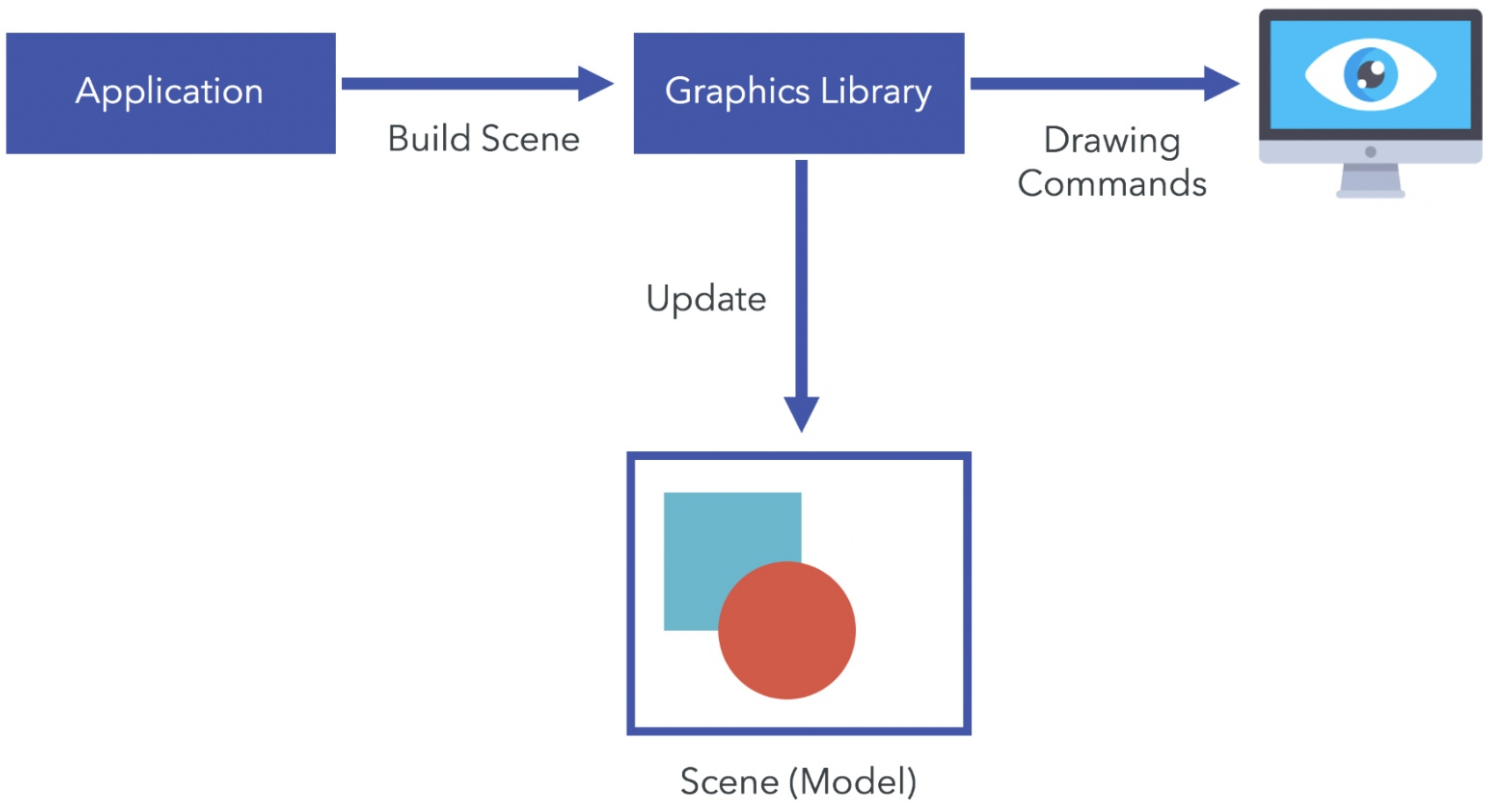
Существуют два принципиально разных режима работы графических API:

- API в режиме сохранения (retained-mode)
- API немедленного режима (immediate-mode)

API в режиме сохранения могут быть проще в использовании, потому что API сама выполняет большую часть работы, например, инициализацию, управление состоянием и очистку. Однако такие API имеют меньшую гибкость и изначально более высокие требования к памяти, поскольку они ориентированы на широкий спектр применения.
API немедленного режима более гибкие и их можно оптимизировать под конкретные задачи.

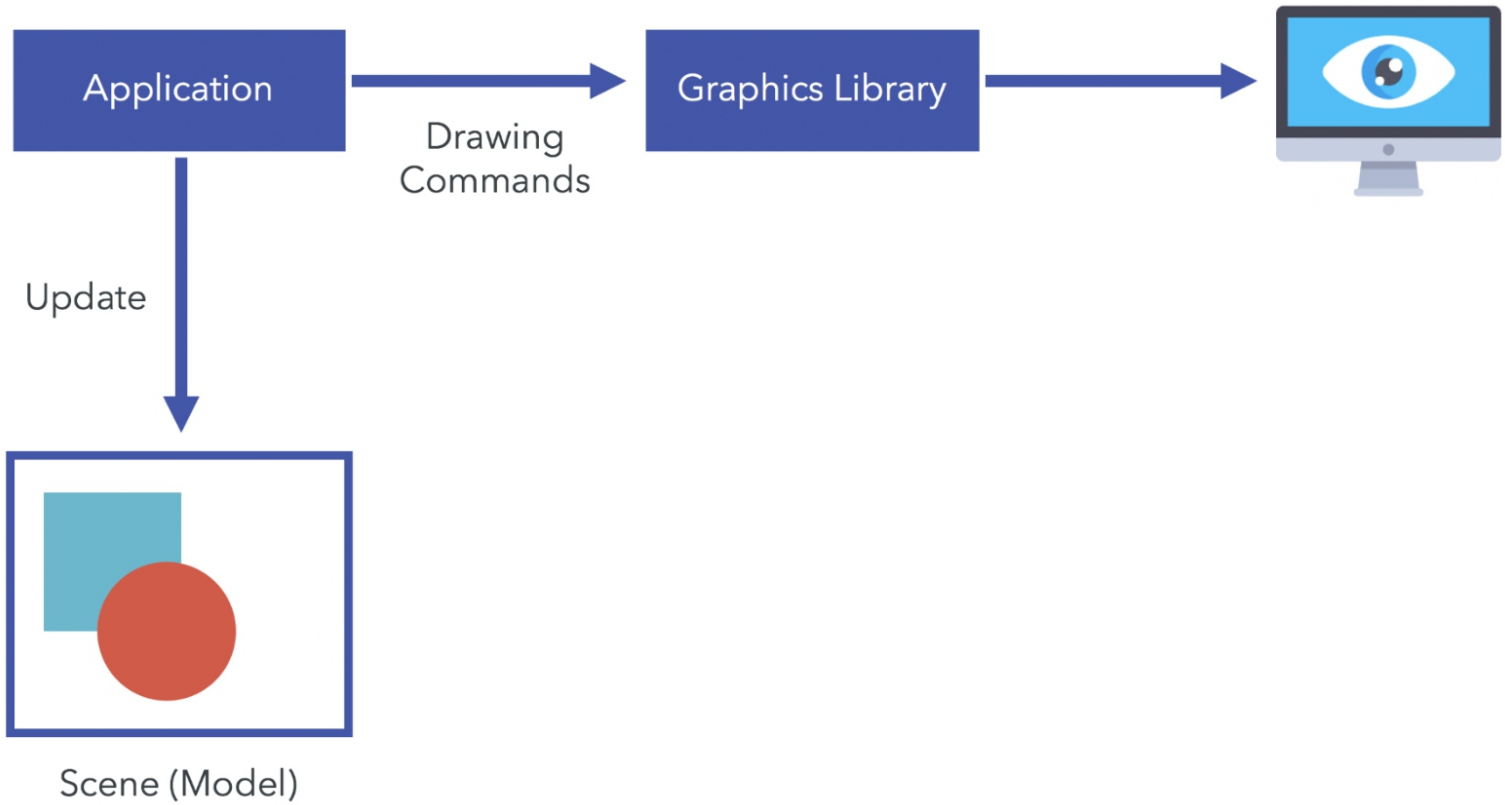
Стандарты разработки графических библиотек

Графическая библиотека, которая предоставляет API в режиме сохранения, является декларативной. Приложение вызывает команды для создания сцены из примитивов, а затем графическая библиотека сама загружает подготовленную сцену в память для отрисовки. Чтобы изменить то, что нарисовано, приложение вызывает команду обновления сцены. Библиотека сама отвечает за управление и перерисовку сцены. В этом режиме приложение может не вызывать команды рисования для отрисовки полной сцены на каждом кадре.



Стандарты разработки графических библиотек

API немедленного режима является процедурным, т.е. приложение само управляет рендерингом. При использовании API немедленного режима сцена полностью перерисовывается на каждом кадре, независимо от того, изменилась ли она. Графическая библиотека, предоставляющая API, не сохраняет внутри себя нарисованную сцену. Этот режим обеспечивает максимальный контроль и гибкость для приложения. Тем не менее, он также может требовать и некоторых накладных расходов.



Координатные представления

При работе с простыми графическими библиотеками требуется, чтобы вся геометрическая информация задавалась в терминах системы координат экрана используемого устройства отображения. Экранные координаты являются целочисленными и соответствуют положениям пикселей. Значения координат пикселей дают номер строки развертки (значение координаты y) и номер столбца (значение координаты x в этой строке развертки). При аппаратном выполнении таких процессов, как обновление экрана, как правило, положения пикселей отсчитываются от левого верхнего угла экрана (левосторонняя система координат). Тогда строкам развертки присваиваются значения от 0 (верхняя строка экрана) до какого-то целочисленного значения y_{max} (нижняя строка экрана), а положения пикселей в каждой строке развертки нумеруются от 0 до x_{max} в направлении слева направо.
Экранные координаты могут быть неудобны для описания изображений, в которых расстояния измеряются в световых годах (при отображении данных, с которыми имеет дело астрономия), или таких, где расстояния измеряются микронами (при выводе изображений топологии интегральных микросхем). Кроме того, для разного оборудования эти системы координат могут быть различны как по направлениям координатных осей, так и по размерам области вывода. Совершенно очевидно, что для отображения трёхмерных сцен такой метод прямо применить нельзя.

Координатные представления

Функции мощной графической библиотеки определяют координаты изображаемых объектов относительно начала отсчёта произвольной декартовой системы координат. Геометрическое описание в этой системе координат может задаваться в любой удобной форме, как с использованием целых чисел, так и чисел с плавающей запятой, без учёта ограничений для отдельных устройств вывода (единственным ограничением является диапазон представления целых или вещественных чисел в компьютере). Возможно связывать эти числа с любыми единицами измерения, выбираемыми в соответствии со спецификой решаемой задачи.

Координатные представления

В общем случае в процессе создания и изображения сцены используется несколько различных декартовых координатных систем.
Во-первых можно задавать формы объектов в отдельной системе координат для каждого объекта. Такие системы координат называются координатами моделирования, локальными или главными координатами. В качестве примера можно составить велосипед, задав каждую из его деталей (колеса, раму, седло, руль, шестеренки, цепь и педали) в отдельной системе координат моделирования. Если оба колеса велосипеда одинаковые по размеру, то в локальной системе координат можно описать только одно колесо.
Задав формы отдельных объектов, можно составить сцену («создать её модель») путём расстановки объектов по соответствующим местам в системе координат сцены, которая называется мировой системой координат. Этот этап подразумевает преобразование отдельных систем координат моделирования в мировую систему координат.
Для описания не слишком важных сцен часть объектов можно вставлять непосредственно в общую структуру сцены в мировых координатах, пропуская этапы задания координат моделирования и преобразования координат моделирования в мировые координаты.

Координатные представления

В некоторых графических библиотеках положения точек можно также задавать с помощью относительных координат. Воспользовавшись этим способом, можно задавать координаты точки относительного последнего положения, к которому обращалась система (его называют текущим положением). Например, если точка с координатами (3, 8) — это последнее положение, к которому обращалась программа, то относительные координаты (2, -1) соответствуют абсолютным координатам (5, 7). В таком случае, перед тем, как задать какие-либо координаты для функций примитивов, используется дополнительная функция, устанавливающая текущее положение. Тогда, чтобы описать такой объект, как набор соединённых между собой прямолинейных отрезков, нужно задать только последовательность относительных координат (смещений) после того, как будет установлено текущее положение.

Графические библиотеки могут предлагать опции, позволяющие задавать положение точки с помощью либо относительных, либо абсолютных координат.

Графические примитивы

Функции графической библиотеки, которые используются для описания различных элементов сцены, называются графическими примитивами или просто примитивами. Примитивы, описывающие геометрию объекта, как правило, называют геометрическими примитивами.
Графическими примитивами могут быть, например, точки, прямые линии, конические сечения, сплайновые кривые и поверхности, плоскости, поверхности второго порядка, простейшие геометрические тела (куб, параллелепипед, сфера, цилиндр, конус, тор и пр.).
Из исходных примитивов можно составить более сложные по геометрической форме графические объекты. Из геометрических примитивов формируются детали, из деталей — объекты, а из объектов — сцены. Таким образом, элементы, из которых составляется графический объект, считаются примитивами для этого объекта.

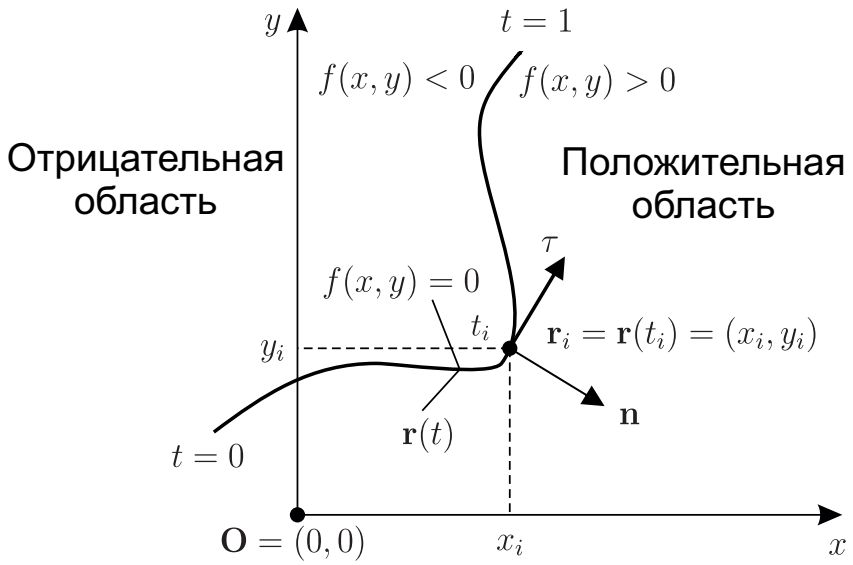
Виды примитивов

Точка — это бесконечно малая величина, размещённая в определённом месте пространства. При её визуализации на растровом устройстве физическая величина точки равна размеру пиксела на плоскости или воксела (единичного элемента изображения) в пространстве.
Линия — это множество соприкасающихся друг с другом в определённом направлении точек. При её отображении на растровом устройстве физическая толщина линии равна размеру пиксела на плоскости или размеру воксела в пространстве. Линия, описываемая линейной зависимостью, называется прямой, нелинейной — кривой. Линии могут быть замкнутыми (окружность, эллипс и пр.) и разомкнутыми (парабола, гипербола и пр.).
Поверхность, описываемая линейной зависимостью, называется плоскостью; нелинейной зависимостью — кривой (криволинейной) поверхностью или просто поверхностью. Поверхность, линия пересечения которой плоскостью во всех возможных направлениях (т.е. линия, точки которой принадлежат одновременно плоскости и поверхности), является замкнутой, также называется замкнутой (например, поверхность сферы, эллипсоида и др.). Выделенная часть поверхности, имеющая ограничения в определённых направлениях, называется ограниченной.

Кривые

Кривые могут использоваться, например, для моделирования геометрии объектов, описания траекторий в анимации, построения графиков данных и функций. Чаще всего встречаются такие кривые, как конические сечения, тригонометрические и показательные функции, распределения вероятности, полиномы общего вида и сплайны.

Кривая линия на плоскости в большинстве случаев описывается либо уравнениями линии в неявной форме: $f(x, y) = 0$, либо уравнениями линии в параметрической форме: $\mathbf{r}(t) = (x(t), y(t))$.



Важным понятием при моделировании является понятие ориентации линии. Используя это понятие, легко формализовать передачу позиционной информации о взаимном расположении точек и линий (слева, справа, внутри, снаружи).

Самый простой способ изображения кривой линии — аппроксимировать её прямолинейными отрезками. Параметрические представления позволяют получить на траектории кривой точки (концы отрезков), находящиеся на одинаковом расстоянии друг от друга.

Конические сечения

В общем случае коническое сечение можно описать с помощью уравнения второго порядка:

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0,$$

где A, B, C, D, E и F — коэффициенты.
Определяя коэффициенты A, B, C, D, E и F можно получить разные конические сечения — параболы, гиперболы, эллипсы, окружности. Это уравнение описывает также вырожденные конические сечения: точки и прямые линии.
Вид конического сечения можно определить, найдя дискриминант:

$$B^2 - 4AC \begin{cases} < 0, & \text{эллипс (или окружность);} \\ = 0, & \text{парабола;} \\ > 0, & \text{гипербола.} \end{cases}$$

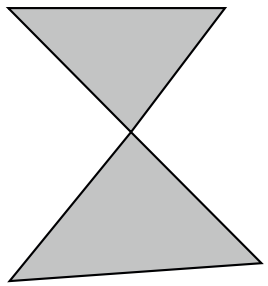
Такие кривые часто используют в прикладной машинной графике, например, для разработки деталей или в чертежных системах.

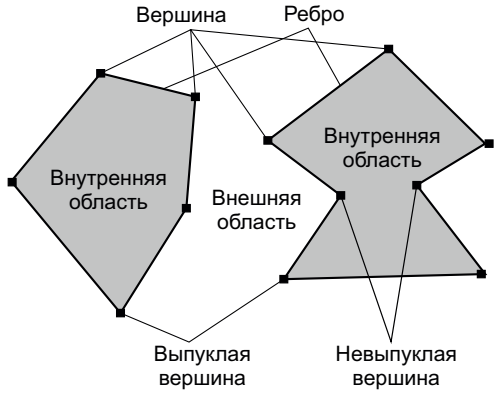
Многоугольники (полигоны)

Многоугольник (полигон) — это плоская фигура, которая задаётся с помощью набора из 3-х или больше точек — вершин, последовательно соединённых прямолинейными отрезками, которые называются ребрами или сторонами многоугольника. Вершины полигона упорядочены циклически вдоль его границы. Полигон является простым, если он не пересекает самого себя. Полигоны с самопересечением всегда можно разделить на несколько простых полигонов.

Примеры многоугольников:

- треугольники,
- прямоугольники,
- восьмиугольники,
- десятиугольники.





В приложениях компьютерной графики возможны такие ситуации, когда не все перечисленные вершины многоугольника лежат строго в одной плоскости. Это может быть связано с ошибкой округления при вычислении значений, либо с ошибкой определения координат вершин или (что более характерно) с аппроксимацией криволинейной поверхности набором многоугольных участков. Один из способов исправления этой ситуации — просто разделить заданную сетку многоугольников на треугольники.

Классификация многоугольников

Многоугольники: слева — выпуклый (convex); справа — невыпуклый (concave)

Внутренним углом многоугольника называется угол внутри границы многоугольника, образованный двумя соседними сторонами. Если все внутренние углы многоугольника меньше или равны $180°$, то многоугольник выпуклый.

Другие эквивалентные определения выпуклого многоугольника:

- Весь многоугольник должен лежать по одну сторону от бесконечной прямой, являющейся продолжением любой из его сторон.
- Если выбрать любые 2 точки внутри выпуклого многоугольника, то прямолинейный отрезок, соединяющей эти точки, также будет полностью находиться внутри многоугольника.
- При движении вдоль границы выпуклой области приходится поворачивать только в одну сторону (только влево или только вправо).

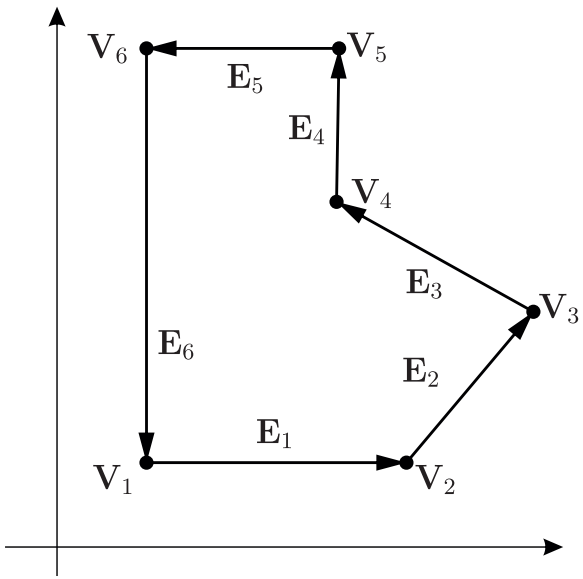
Классификация многоугольников

Невыпуклые многоугольники создают дополнительные проблемы. Реализация алгоритмов закрашивания и других графических процедур для невыпуклых многоугольников намного сложнее, поэтому, как правило, перед обработкой эффективнее разделить невыпуклый многоугольник на набор выпуклых. Как и другие алгоритмы предварительной обработки многоугольников, функции разделения невыпуклого многоугольника часто не входят в графическую библиотеку.
Для некоторых графических библиотек, в том числе и OpenGL, необходимо, чтобы все закрашиваемые многоугольники были выпуклыми. Кроме того, в некоторых системах допускается только закрашивание треугольных областей, что значительно облегчает многие алгоритмы создания изображений.

Распознавание невыпуклых многоугольников

Признаки невыпуклого многоугольника, которые можно положить в основу алгоритма распознавания невыпуклых многоугольников:

- по крайней мере один внутренний угол больше чем 180°;
- продолжения некоторых сторон невыпуклого многоугольника будут пересекать другие ребра, а вершины находиться по разные стороны от такого продолжения;
- некоторые пары внутренних точек дадут отрезки, пересекающие границы многоугольника;
- для выпуклого многоугольника все векторные произведения соседних сторон будут одного знака (положительные или отрицательные). В противном случае, многоугольник — невыпуклый.



$$(\mathbf{E}_1 \times \mathbf{E}_2)_z > 0$$

$$(\mathbf{E}_2 \times \mathbf{E}_3)_z > 0$$

$$(\mathbf{E}_3 \times \mathbf{E}_4)_z < 0$$

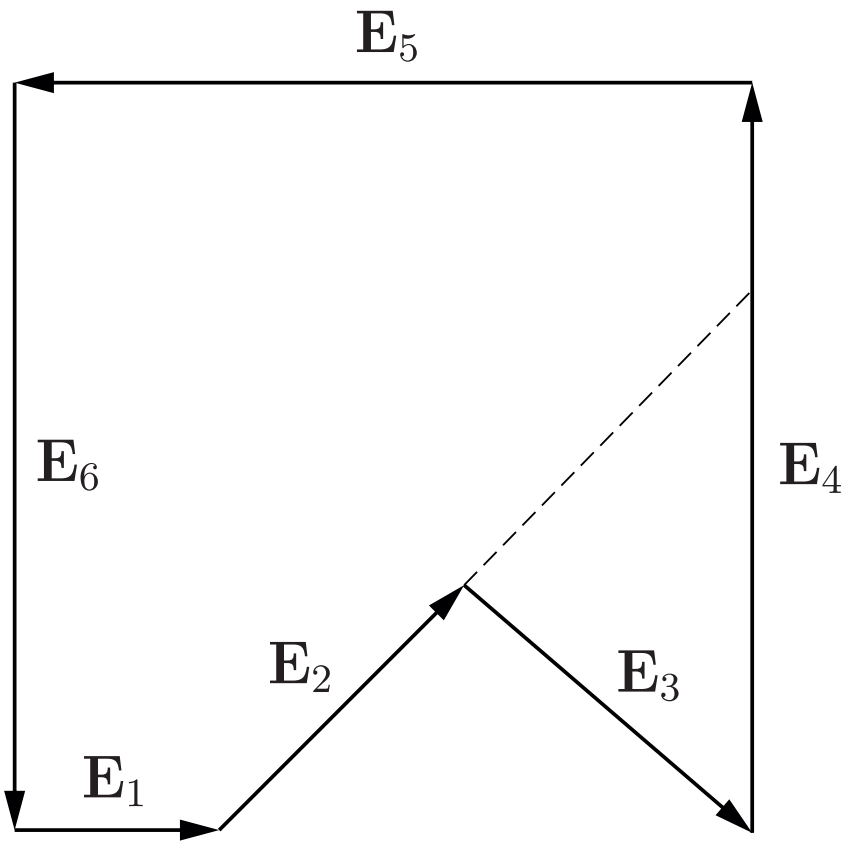
$$(\mathbf{E}_4 \times \mathbf{E}_5)_z > 0$$

$$(\mathbf{E}_5 \times \mathbf{E}_6)_z > 0$$

$$(\mathbf{E}_6 \times \mathbf{E}_1)_z > 0$$

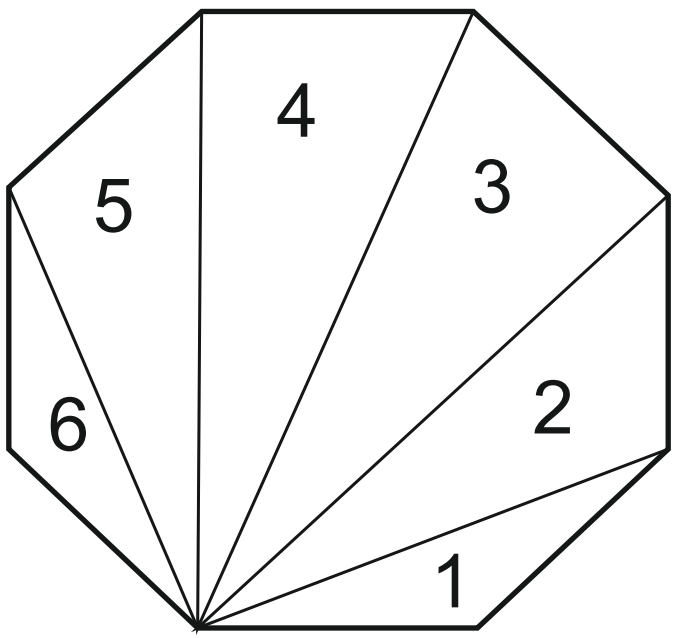
Деление невыпуклых многоугольников

Если при обходе вершин против часовой стрелки, проекция векторного произведения сторон, образуемых этими вершинами, на ось z становится отрицательной, то многоугольник невыпуклый и его можно разделить, продолжив первый вектор стороны из этой пары векторов, произведение которых рассматривается.

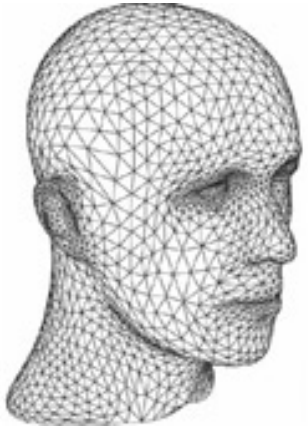


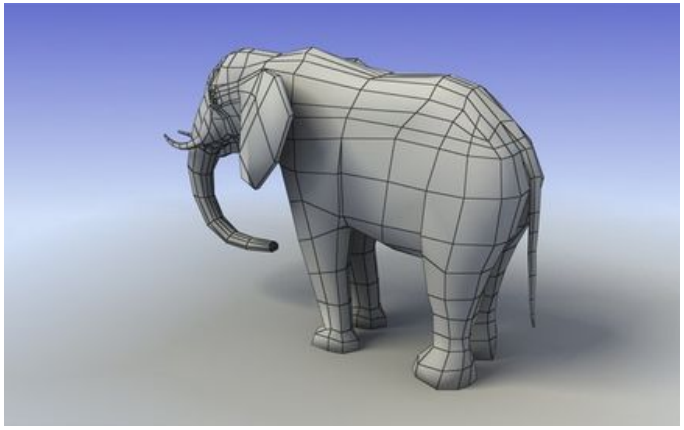
Разделение выпуклого многоугольника на набор треугольников

Если дан список вершин выпуклого многоугольника, его можно преобразовать в набор треугольников. Для этого сначала определяется любая последовательность из трех идущих подряд вершин, которые образуют новый многоугольник (треугольник). После этого, из исходного списка вершин удаляется средняя вершина треугольника. Затем та же процедура выполняется с измененным списком вершин, и вырезается еще один треугольник. Формирование треугольников продолжается до тех пор, пока исходный список вершин многоугольника не уменьшится до трех, которые и определяют координаты вершин последнего треугольника этого набора.



Таблицы многоугольников





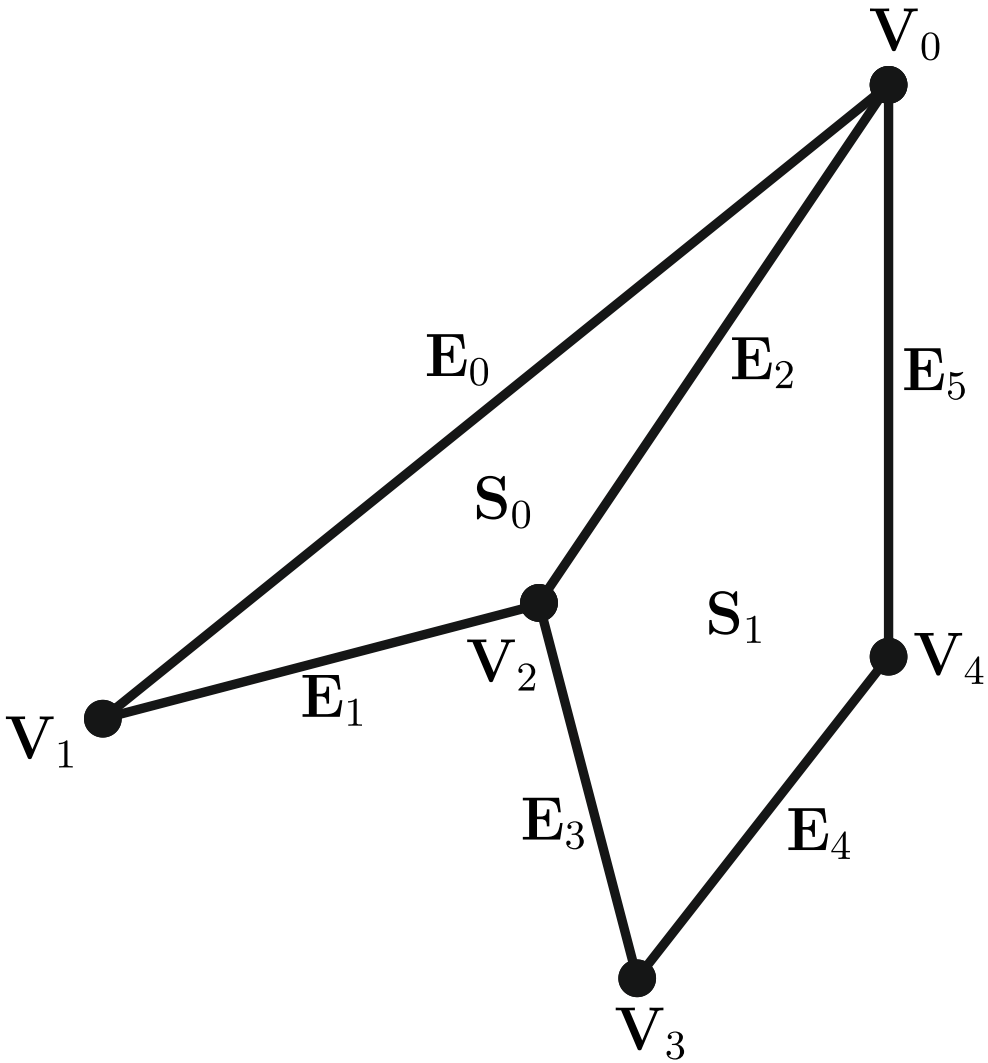
Как правило, объекты сцены описываются как наборы многоугольных граней поверхностей. Графические библиотеки часто содержат функции отрисовки поверхности в виде сетки многоугольных участков.
Описание каждого объекта включает геометрическую информацию и информацию об атрибутах. К геометрической информации относятся данные о координатах вершин и пространственной ориентации. К атрибутам объекта относятся, например, величины, определяющие степень прозрачности объекта, отражающую способность его поверхности (цвет) и текстурные характеристики.
Геометрическую информацию об объектах сцены удобно распределить по таблицам вершин (vertices), ребер (edges) и граней (faces).

Таблицы многоугольников

Координаты всех вершин объекта записываются в таблицу вершин.

Номер (индекс) вершины	Координаты вершины
0	x_0, y_0, z_0
1	x_1, y_1, z_1
2	x_2, y_2, z_2
3	x_3, y_3, z_3
4	x_4, y_4, z_4

Таблица: Таблица вершин

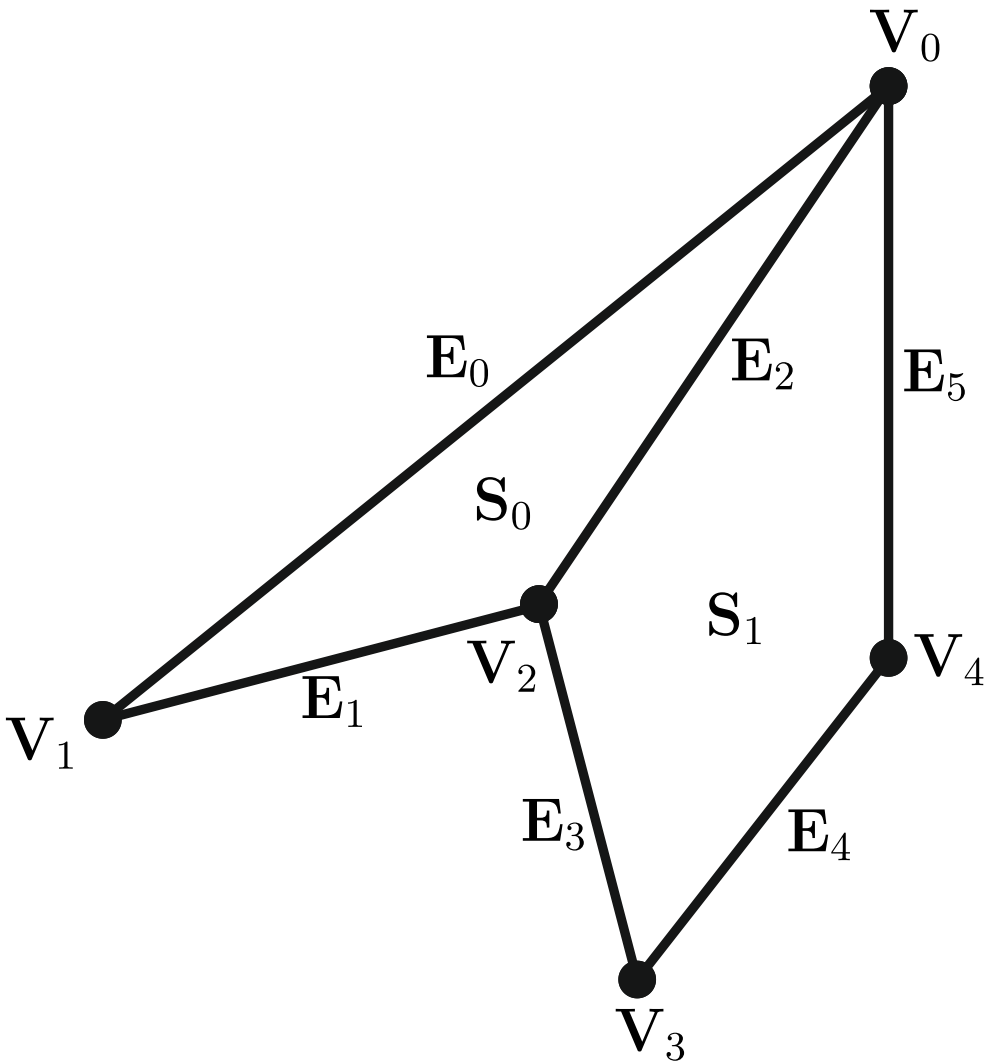


Таблицы многоугольников

Таблица ребер используется для построения каркасного вида модели. В ней содержатся ссылки на таблицу вершин, позволяющие определить вершины, принадлежащие каждому ребру многоугольника.

Номер (индекс) ребра	Номера (индексы) вершин
0	0, 1
1	1, 2
2	2, 0
3	2, 3
4	3, 4
5	4, 0

Таблица: Таблица ребер

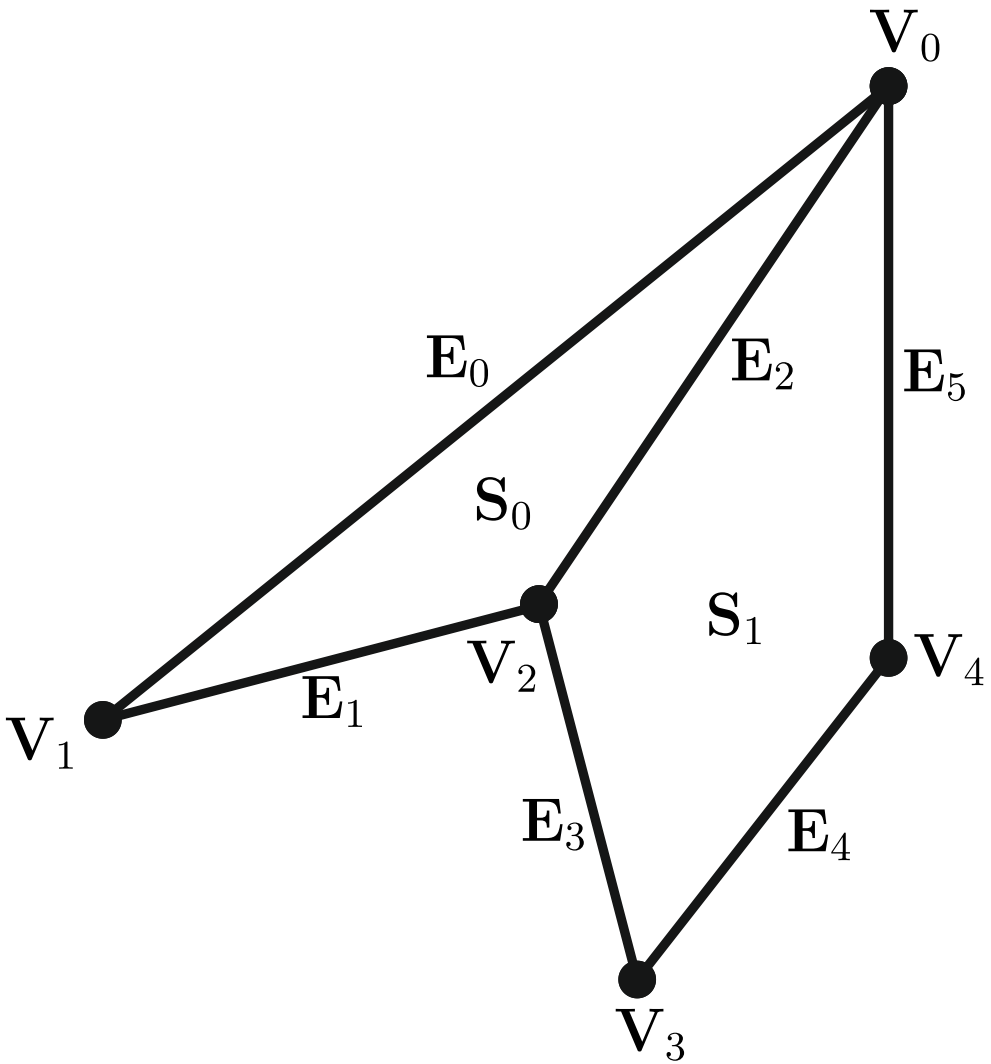


Таблицы многоугольников

Таблица граней используется для построения полноцветного вида модели. В ней содержатся ссылки на таблицу вершин, позволяющие определить вершины, принадлежащие каждой грани многоугольника.

Номер (индекс) грани	Номера (индексы) вершин
0	0, 1, 2
1	0, 2, 3, 4

Таблица: Таблица граней

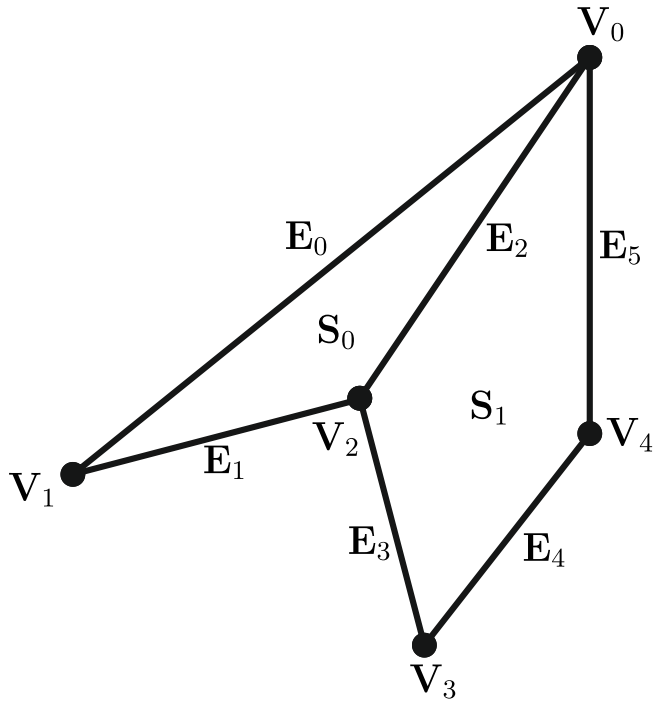


Таблицы многоугольников

Далее можно сформировать и другие таблицы:

- Расширенную таблицу ребер, включающую указатели на таблицу граней, чтобы быстрее определить общие стороны многоугольников. Это особенно удобно для процедур визуализации, в которых затенение поверхности при переходе через ребро от одного многоугольника к другому должно изменяться плавно.

Номер ребра	Номера вершин	Номера граней
0	0, 1	0
1	1, 2	0
2	2, 0	0, 1
3	2, 3	1
4	3, 4	1
5	4, 0	1



- Для более быстрого доступа к данным можно расширить таблицу вершин, дополнив её ссылками на соответствующие грани.
- Расширенную таблицу граней с указанием отдельных поверхностей в сцене, которые они формируют.

Уравнения плоскости

Каждый многоугольник сцены лежит в какой-то бесконечной плоскости. Общий вид уравнений плоскости:

$$Ax + By + Cz + D = 0,$$

где (x, y, z) — произвольная точка на этой плоскости, а коэффициенты A, B, C и D — константы, описывающие пространственные свойства плоскости. Значения констант A, B, C и D можно найти, решив систему из трёх уравнений плоскости, подставив в неё координаты трёх неколлинеарных точек, лежащих в этой плоскости. Для этого можно выбрать три вершины многоугольника (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3) и решить следующую систему линейных уравнений плоскости относительно отношений A/D, B/D и C/D:

$$(A/D)\, x_k + (B/D)\, y_k + (C/D)\, z_k = -1, \qquad k = 1, 2, 3.$$

Выражения для коэффициентов плоскости:

$$\begin{aligned}
A &= y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2), \\
B &= z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2), \\
C &= x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2), \\
D &= -x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_3 z_1 - y_1 z_3) - x_3 (y_1 z_2 - y_2 z_1).
\end{aligned}$$

Эти выражения справедливы и в случае если $D = 0$.

Передние и задние грани многоугольника

Поскольку обычно многоугольные поверхности, с которыми приходится иметь дело в графических приложениях, представляют собой внешнюю оболочку объекта, необходимо различать две стороны каждой поверхности. Определение положения точек в пространстве относительно передней и задней сторон многоугольника является основной задачей многих графических алгоритмов, например, алгоритма определения видимых частей объекта.

Чтобы определить положения точек в пространстве относительно многоугольных граней объекта, можно воспользоваться уравнением плоскости. Для любой точки (x, y, z):

- если $Ax + By + Cz + D < 0$, то точка (x, y, z) находится за плоскостью;
- если $Ax + By + Cz + D > 0$, то точка (x, y, z) находится перед плоскостью;

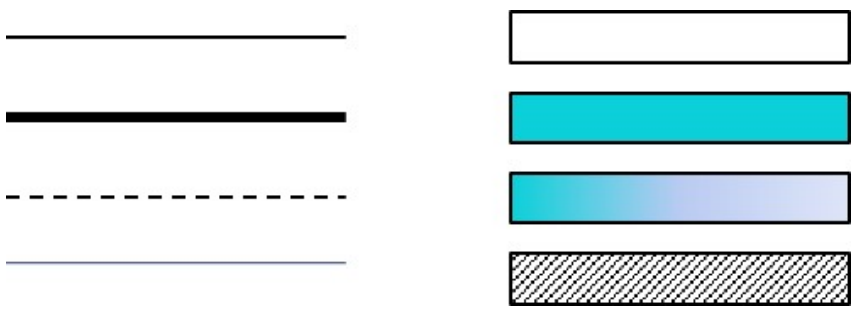
Здесь параметры плоскости A, B, C и D вычисляются с помощью координат вершин, выбранных строго против часовой стрелки, если смотреть на внешнюю поверхность.

Ориентацию многоугольной поверхности в пространстве можно описать с помощью вектора нормали к плоскости многоугольника, который имеет компоненты: (A, B, C) — внешняя нормаль.

Компоненты вектора нормали можно также найти, вычисляя векторное произведение векторов-сторон многоугольника.

Атрибуты графических примитивов

В компьютерной графике принято разделять информацию, определяющую форму объекта, и информацию, характеризующую вид объекта на экране. Красная сплошная линия и зеленая штриховая линия формируются на основании одного и того же типа примитива, но выглядят совершенно по-разному.
Атрибут — это параметр, который влияет на способ изображения примитива.
Цвет — это основной атрибут всех примитивов.



Цвет

Одно из представлений цвета в компьютерной графике основывается на предположении, что любой цвет можно рассматривать как взвешенную сумму трёх основных (первичных) цветов (*аддитивная цветовая модель, additive color model*). Цветовой поток, который представляет собой наложение трёх первичных цветов, наши глаза воспринимают как один смешанный цвет.

При использовании такой модели можно, варьируя интенсивность излучения каждого из первичных цветов, добиться довольно близкого приближения к желаемому цвету.

В цветных мониторах в качестве первичных используются красный, зелёный и синий цвета (RGB).

Цветовые опции могут задаваться численно, выбираться из меню или с помощью ползунков. На мониторе эти коды цвета преобразуются в настройки уровня интенсивности свечения пикселей.

Прозрачность поверхности

Поверхность называется прозрачной, если видны предметы, находящиеся за ней. Если предметы, расположенные за поверхностью, не видны, то поверхность является непрозрачной. В общем случае поверхность может быть полупрозрачной, т.е. и отражать, и пропускать свет (например, матовое стекло, определённые пластические материалы).
Интенсивность I_d пронесённую от фонового объекта через прозрачную поверхность, можно объединить с отраженной от поверхности интенсивностью I_s, используя коэффициент прозрачности α. Параметру α присваивается значение между 0.0 и 1.0, и он определяет, какое количество фонового света было пропущено. Суммарная интенсивность поверхности вычисляется как

$$I = (1 - \alpha)I_s + \alpha I_d.$$

Член $(1 - \alpha)$ — это коэффициент непрозрачности. В графических библиотеках он носит название альфа-канала и может добавляться к первичным составляющим цвета (модель RGBA). Например, если коэффициент непрозрачности имеет значение 0.7, то 30% фонового света объединяется с 70% отраженной освещённости поверхности.
Описанную процедуру можно использовать для вычисления суммарного цвета, порождённого любым числом прозрачных и непрозрачных объектов, если обрабатывать поверхности в порядке уменьшения глубины (от заднего плана к переднему).

Графическая библиотека OpenGL

Январь 1992 г. — разработана первая версия пакета OpenGL 1.0 (Open Graphics Library - открытая графическая библиотека). Термин «открытый» означает независимый от производителей. В том же году был создан Совет по развитию архитектуры OpenGL (OpenGL Architectural Review Board (ARB)), в первоначальный состав которой вошли такие компании, как Silicon Graphics, Inc.(SGI), Compaq, Digital Equipment Corporation (DEC), IBM, Intel и Microsoft. Вскоре, другие компании, такие как Hewlett-Packard, Sun Microsystems, Evans & Sutherland и Intergraph вошли в этот Совет. OpenGL ARB является органом стандартизации, которая разрабатывает, управляет и производит спецификацию OpenGL и теперь является частью Khronos Group, которая является некоммерческим консорциумом компаний, который осуществляет надзор за развитием, публикацией и продвижением свободных открытых стандартов параллельных вычислений, графических и динамических мультимедийных технологий для широкого диапазона платформ и устройств.
На данный момент существует 20 изданий стандарта OpenGL (последняя из которых OpenGL 4.6 вышла в июле 2017). Для использования возможностей новых версий требуется обновление драйверов и аппаратного обеспечения.

Графическая библиотека OpenGL

Разработана специально для эффективной обработки трёхмерных данных, но может работать и с описаниями двумерных сцен как с частным случаем трёхмерного изображения, где все значения координаты z равны нулю.
Программы, написанные с помощью OpenGL, можно переносить практически на любые аппаратные (персональный компьютер, графическая станция или суперкомпьютер) и программные платформы (Unix, Linux, SunOS, IRIX, Windows, Mac OS X, и другие), получая при этом одинаковый результат. Существуют привязки пакета OpenGL ко всем основным языкам C(C++), C#, Java, Visual Basic и Fortran. Общие принципы использования OpenGL в любой системе программирования одинаковы.
OpenGL используется в приложениях моделирования и визуализации, применяется для вывода графических данных в САПР, дизайнерских программах, а также компьютерных игр (Quake).
Стандарт OpenGL поддерживает функциональный интерфейс.
Open Inventor — объектно-ориентированная библиотека классов для описания сцен, которые нужно изобразить с помощью функций OpenGL.

Графическая библиотека DirectX

1995 г. — компания Microsoft представила первую версию библиотеки DirectX (тогда она называлась Game SDK). Все права на доработку/изменение DirectX принадлежат Microsoft.
DirectX предназначен только для платформ под управлением ОС Windows.
Стандарт DirectX основан на выборе интерфейсов Component Object Model (компонентная модель объектов), а объекты COM могут описываться практически любыми языками программирования, например C++, C# Visual Basic .NET.
Стандарт DirectX включает в себя модули поддержки:

- программирования двухмерной графики (модуль DirectDraw);
- создания трёхмерной графики (модуль Direct3D);
- работы со звуками и музыкой (модули DirectSound и DirectMusic);
- поддержки устройств ввода (модуль DirectInput);
- разработки сетевых игр (модуль DirectPlay);
- проигрывание мультимедийных потоков данных (модуль DirectShow);
- инсталляции компонентов DirectX (DirectSetup).

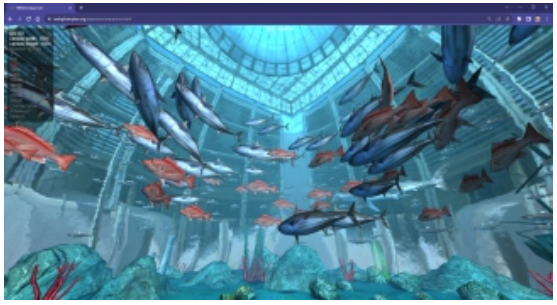
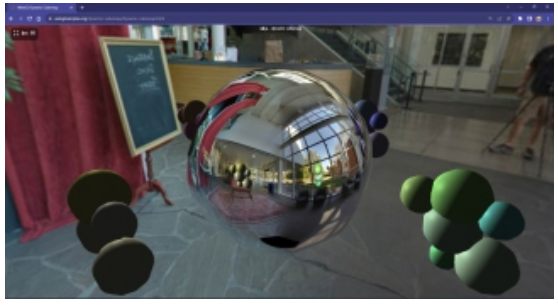
Используется:

- при разработке графических интерфейсов Windows;
- в мультимедийных приложениях, играх, тренажерах и других сложных программах, предусматривающих активное взаимодействие с пользователем;
- ...

OpenGL против DirectX

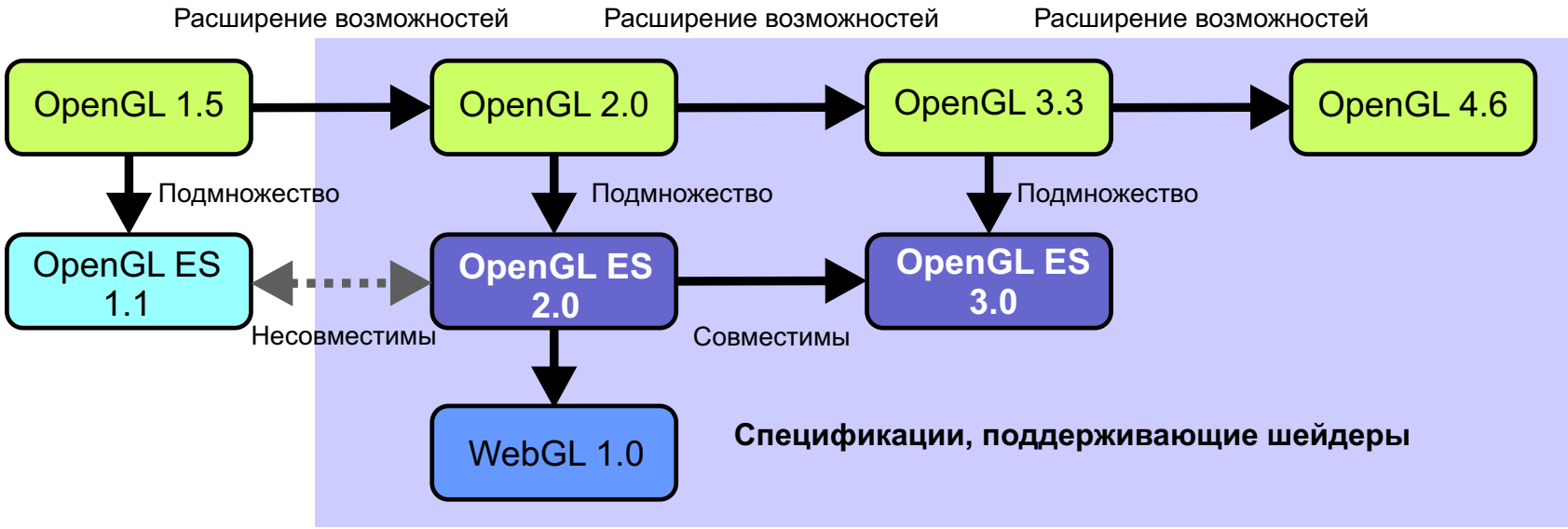
- Сегодня все ведущие производители видеокарт предоставляют OpenGL и DirectX драйвера для своей продукции, позволяющие программе задействовать аппаратную поддержку.
- В настоящее время подавляющее большинство прикладных программ, работающих с трёхмерными объектами, опираются на одну из двух типовых библиотек – OpenGL или DirectX.
- OpenGL в основном используется в области профессиональных графических решений, но имеет все шансы занять главенствующее положение и в разработке обычных графических приложений.
- Большинство компьютерных игр (не говоря о графическом интерфейсе ОС Windows) используют исключительно библиотеку от Microsoft. OpenGL является достаточно быстрой библиотекой для вывода трехмерной графики даже на слабых системах. Основанная на технологии COM Direct3D значительно медленнее.
- В отличие от Direct3D, OpenGL очень проста в изучении и интуитивно понятна.
- Совместное использование графических модулей DirectX и OpenGL невозможно, но ничто не мешает использовать другие модули, такие как DirectInput.

Графическая библиотека WebGL


Благодаря росту производительности персональных компьютеров и расширению возможностей браузеров стало возможным создание и отображение трёхмерной графики с применением веб-технологий. WebGL — это технология, позволяющая рисовать, отображать и взаимодействовать со сложной, интерактивной трёхмерной компьютерной графикой в веб-браузерах. WebGL позволяет использовать в браузере преимущества аппаратного ускорения трёхмерной графики и не требует установки специализированных расширений или библиотек. Благодаря этому, одна и та же программа успешно будет выполняться на самых разных устройствах, таких как смартфоны, планшетные компьютеры и игровые консоли. С помощью WebGL, разработчики могут создавать трёхмерные игры и использовать трёхмерную графику для визуализации различной информации из Интернета.

Графическая библиотека WebGL



Спецификация WebGL основана на стандарте OpenGL. Название «WebGL» можно интерпретировать как «OpenGL для веб-браузеров». WebGL является дальнейшим развитием версии OpenGL ES (for Embedded Systems — для встраиваемых систем, таких как смартфоны и игровые консоли). Эта версия создана в 2003–2004 годах и затем была обновлена в 2007 году (ES 2.0) и в 2012 (ES 3.0). В 2009 году консорциум Khronos Group учредил рабочую группу WebGL и запустил процесс стандартизации WebGL на основе OpenGL ES 2.0. В марте 2011 года под его эгидой была выпущена первая версия WebGL. С 27 февраля 2017 года доступна WebGL 2.0. Она построена на основе OpenGL ES 3.0.

Графическая библиотека WebGL

По сравнению с другими технологиями (такими как Java 3D, Flash и Unity Web Player Plugin), WebGL имеет несколько преимуществ:

Программирование на JavaScript. Работа с JavaScript позволяет получить доступ к модели DOM и легко интегрируют приложения WebGL с другими библиотеками JavaScript, такими как jQuery, React и Angular.

Автоматическое управление памятью. В отличие, например, от OpenGL, где вручную осуществляется выделение и освобождение памяти, WebGL следует правилам области видимости переменных JavaScript и автоматического управления памятью. Это значительно упрощает программирование и сокращает объём кода. В конечном итоге это упрощает понимание логики приложения.

Доступность. Веб-браузеры с возможностями JavaScript устанавливаются в смартфоны и планшеты по умолчанию. Благодаря этому, одна и та же программа успешно будет выполняться на самых разных устройствах.

Производительность. WebGL позволяет использовать в браузере преимущества аппаратного ускорения трёхмерной графики.

Отсутствие компиляции. WebGL написан на JavaScript; следовательно, нет необходимости в компиляции кода перед его выполнением в веб-браузере.

WebGL реализует аппаратный рендеринг.

WebGL реализует локальный рендеринг: элементы 3D сцены загружаются с сервера и визуализируются на компьютере клиента.

WebGL как и OpenGL поддерживает немедленный режим работы.

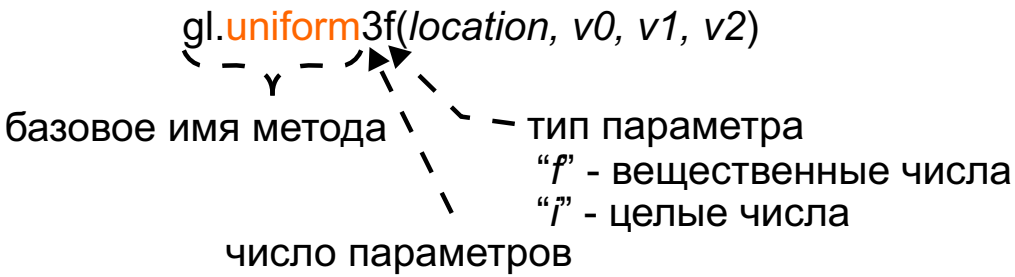
Контекст WebGL

In WebGL 2, you'd simply attain the context with a single line, as follows: Для получения контекста WebGL 2 требуется выполнить команду:

```
const context = canvas.getContext('webgl2');
```

В большинстве программ WebGL-контекст называется `gl`, потому что методы и значения, относящиеся к OpenGL, обычно имеют префикс `gl`. Следование этой конвенции делает JavaScript-код более похожим на OpenGL-программу.

Правила именования методов WebGL



Имена многих OpenGL- и WebGL-методов включают сведения о типах их аргументов. Если метод может принимать разное количество аргументов разных типов, к его имени добавляется соответствующий суффикс. Число в нём указывает количество аргументов (от 1 до 4), а буква — их тип («`f`» означает `float`, то есть вещественное число или «`i`»(`integer`) для целых чисел). Например, метод `gl.uniform4f()` ожидает четыре числа с плавающей точкой, а метод `gl.uniform3i()` — три целых числа. Для обозначения всех методов, от `gl.uniform1f()` до `gl.uniform4f()`, можно использовать такую нотацию: `gl.uniform[1234]f()`, где квадратные скобки указывают, что на их месте должно быть указано одно из перечисленных чисел.

Многие методы также могут принимать массив вместо отдельных аргументов, на что указывает буква «`v`» (`vector`). Так, метод `gl.uniform4fv()` принимает массив с четырьмя числами типа `float`:

```
const lightPosition = new Float32Array([1.0, 2.0, 3.0, 1.0]);
gl.uniform4fv(u_lPosition, lightPosition);
```

Указать цвет для очистки области рисования <canvas>

Одно из первых действий при работе с WebGL-контекстом — заполнение элемента <canvas> сплошным цветом для подготовки к рисованию. Для этого сначала нужно задать цвет:

`gl.clearColor(red, green, blue, alpha)`		
Устанавливает цвет для очистки (заливки) области рисования.		
Параметры	`red`	Определяет интенсивность красной составляющей цвета (от 0.0 до 1.0).
	`green`	Определяет интенсивность зелёной составляющей цвета (от 0.0 до 1.0).
	`blue`	Определяет интенсивность синей составляющей цвета (от 0.0 до 1.0).
	`alpha`	Определяет значение альфа-канала (прозрачность) цвета (от 0.0 до 1.0). Значение 0.0 соответствует полностью прозрачному цвету, значение 1.0 – полностью непрозрачному.
	Любые значения меньше 0.0 и больше 1.0 в этих параметрах усекаются до 0.0 и 1.0, соответственно.	
Возвращаемое значение	нет	

Цвет сохраняется в системе WebGL и не изменяется, пока не будет указан другой цвет вызовом `gl.clearColor()`.

Очистить <canvas>

`gl.clear(buffer)`		
Очищает указанный буфер предварительно определёнными значениями.		
Параметры	`buffer`	Определяет очищаемый буфер. С помощью оператора | (поразрядная операция «ИЛИ») можно указать несколько буферов. `gl.COLOR_BUFFER_BIT` Буфер цвета. `gl.DEPTH_BUFFER_BIT` Буфер глубины. `gl.STENCIL_BUFFER_BIT` Буфер трафарета.
Возвращаемое значение	нет	
Ошибки		Аргумент `buffer` имеет значение, отличное от любого из трёх, указанных выше.

Значения по умолчанию для очистки буферов и соответствующие методы установки значений очистки:

Буфер	Значение по умолчанию	Метод установки
Буфер цвета	(0.0,0.0,0.0,0.0)	`gl.clearColor(red, green, blue, alpha)`
Буфер глубины	1.0	`gl.clearDepth(depth)`
Буфер трафарета	0	`gl.clearStencil(s)`

Буфер кадра

Буфер кадра (*framebuffer*) — это память, которая содержит информацию, необходимую для отображения окончательного изображения на дисплее. Физическая память, которая используется для буфера кадра, может быть расположена в разных местах. Для простой графической системы буфер кадра может быть выделен как часть обычной основной памяти, но современные графические системы обычно имеют буфер кадра, который выделяется в специальной быстрой графической памяти на графическом процессоре или, возможно, на отдельном чипе, очень близком к графическому процессору.
Буфер кадра обычно состоит как минимум из трёх разных буферов:

- буфер цвета (color buffer)
- буфер глубины (Z-буфер)
- буфер трафарета (stencil buffer)

Буфер цвета

Буфер цвета представляет собой прямоугольный массив памяти, который содержит цвет в формате RGB или RGBA для каждого пикселя на экране. Общее количество бит, доступных для представления цвета одного пикселя, называется глубиной цвета. Примеры глубин цвета:

- 16 бит на пиксель
- 24 бит на пиксель
- 32 бита на пиксель

Буфер цвета с глубиной 16 бит на пиксель часто используется в небольших устройствах, таких как некоторые простые мобильные телефоны. Когда у вас есть 16 бит на пиксель, распределение между цветами составляет: 5 бит для красного, 6 бит для зелёного, 5 бит для синего. Альфа-канал отсутствует. Этот формат часто упоминается как RGB565. Причиной выбора зелёного цвета для дополнительного бита является то, что человеческий глаз наиболее чувствителен к зелёному свету. Этот режим даёт $2^{16} = 65536$ цветов.
Буфер цвета с глубиной 24 бит на пиксель выделяет 8 бит для красного, 8 бит для зеленого и 8 бит для синего. Он даёт более 16 миллионов цветов и отсутствие альфа-канала в буфере цвета.
Буфер цвета с глубиной 32 бит на пиксель обычно имеет такое же распределение бит, как 24-битный буфер цвета плюс 8 бит, выделенных для альфа-канала.

Z-буфер

Когда 3D-сцена сцена отображается на 2D-устройстве, буфер цвета содержит только цвета объектов на сцене. В то же время, некоторые объекты в 3D-сцене могут быть закрыты другими объектами. Чтобы фрагменты, принадлежащие к скрытым объектам, не были видны, используется буфер глубины или Z-буфер. Z-буфер имеет такое же количество элементов, сколько пикселей в цветовом буфере. Для каждого пикселя буфер глубины сохраняет расстояние от наблюдателя до ближайшего к нему примитива.
Для этого координата z каждого фрагмента сравнивается с текущим значением z в буфере глубины для того же пикселя. Если значение z для нового фрагмента меньше, чем значение z в буфере глубины, то новый фрагмент ближе к наблюдателю, чем фрагмент, который был ранее сохранен в буфере цвета, и новый фрагмент берется вместо старого. Если значение z нового фрагмента больше, чем значение в Z-буфере, то новый фрагмент оказывается дальше фрагмента, который используется в настоящее время, и поэтому этот новый фрагмент отбрасывается.
Чтобы активизировать Z-буфер в WebGL, требуется выполнить следующую команду:

```
gl.enable(gl.DEPTH_TEST);
```

Буфер трафарета

Буфер трафарета можно использовать для управления тем, что должно отрисовываться в буфере цвета. Он может быть использован, например, для обработки теней.

Смешивание цветов в WebGL

Во многих приложениях требуется сочетать цвета накладывающихся друг на друга объектов или смешивать цвет объекта с цветом фона, например, при моделировании эффектов прозрачности. Данные функции носят название *функций смешивания цветов*. В пакете WebGL цвета двух объектов можно смешать, сначала загрузив в буфер кадра один цвет, а затем объединив цвет второго объекта с цветом из буфера кадра. Чтобы применить смешивание цветов в приложении, сначала нужно активизировать эту возможность пакета WebGL с помощью такой функции:

```
gl.enable (gl.BLEND);
```

Чтобы отключить стандартные процедуры смешивания цветов, в WebGL используется функция

```
gl.disable (gl.BLEND);
```

Если возможность смешивания цветов не активизирована, цвет объекта просто заменит цвет, записанный в буфере кадра в положении этого объекта.

Смешивание цветов в WebGL

Цвета можно смешивать несколькими различными способами, в зависимости от эффекта, которого требуется достичь, с помощью задания двух наборов *коэффициентов смешивания*. Один набор коэффициентов смешивания задаётся для текущего объекта в буфере кадра («получатель» (destination)), а второй набор коэффициентов — для нового объекта («источник»(source)). Новый смешанный цвет, который затем загружается в буфер кадра, находится как

$$(S_r R_s + D_r R_d, S_g G_s + D_g G_d, S_b B_s + D_b B_d, S_a A_s + D_a A_d),$$

где цветовые компоненты RGBA источника — (R_s, G_s, B_s, A_s), цветовые компоненты получателя — (R_d, G_d, B_d, A_d), коэффициенты смешивания источника — (S_r, S_g, S_b, S_a), а коэффициенты смешивания получателя — (D_r, D_g, D_b, D_a). Найденные значения компонентов комбинированного цвета должны попадать в диапазон от 0.0 до 1.0. Любой сумме, превышающей 1.0, присваивается значение 1.0, меньшей 0.0 — 0.0. Значения коэффициентов смешивания выбираются с помощью следующей функции WebGL:

```
gl.blendFunc (sFactor, dFactor);
```

Коэффициентам источника `sFactor` и получателя `dFactor` присваиваются символьные константы WebGL, задающие предопределённый набор из четырёх коэффициентов смешивания.

Смешивание цветов в WebGL

Например, константа `gl.ZERO` даёт коэффициенты смешивания $(0.0, 0.0, 0.0, 0.0)$, а константа `gl.ONE` даёт набор $(1.0, 1.0, 1.0, 1.0)$. Можно присвоить всем четырём коэффициентам смешивания либо значение альфа получателя, либо значение альфа источника, что делается с помощью константы `gl.DST_ALPHA` или `gl.SRC_ALPHA`. К числу остальных констант `OpenGL`, с помощью которых задаются коэффициенты смешивания, относятся `gl.ONE_MINUS_DST_ALPHA`, `gl.ONE_MINUS_SRC_ALPHA`, `gl.DST_COLOR` и `gl.SRC_COLOR`. По умолчанию параметру `sFactor` присваивается значение `gl.ONE`, а значение параметра `dFactor` по умолчанию равно `gl.ZERO`. Следовательно, по умолчанию значения коэффициентов смешивания приводят к тому, что новые цветовые значения заменяют текущие значения в буфере кадра. Предположим, что нужно нарисовать рисунок, составленный из трёх полупрозрачных поверхностей на залитом фоне, частично перекрывающихся. Пусть самая дальняя поверхность пропускает 80% цвета за ней, следующая пропускает 40%, а ближняя — 90%. Для получения этого изображения сначала рисуется фон с факторами влияния источника и получателя по умолчанию, затем меняем коэффициенты смешивания на `gl.SRC_ALPHA` (для источника) и `gl.ONE_MINUS_SRC_ALPHA` (для получателя). Теперь рисуем дальнюю поверхность с $A = 0.2$, затем промежуточную поверхность с $A = 0.6$ и, наконец, ближнюю поверхность с $A = 0.1$.

Область просмотра

В WebGL область просмотра по умолчанию занимает весь холст. Для изменения области просмотра используется метод

`gl.viewport(x, y, width, height)`

Определяет область рисования.

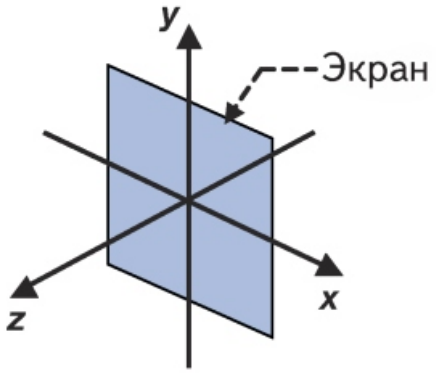
В WebGL параметры `x` и `y` задаются в системе координат элемента <canvas>.

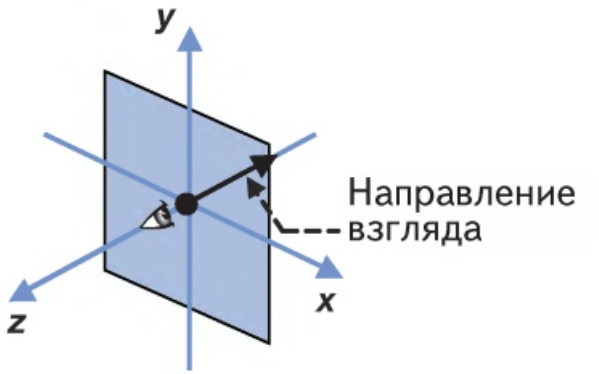
Параметры	`x,y`	Координаты нижнего левого угла прямоугольника области рисования (в пикселях).
	`width`	Ширина области рисования (в пикселях).
	`height`	Высота области рисования (в пикселях).
Возвращаемое значение	нет	
Ошибки	нет	

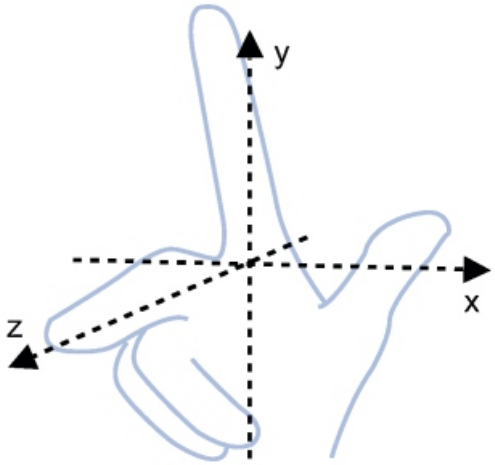
Примеры:

```
// область просмотра во весь холст (режим по умолчанию):
gl.viewport(0, 0, canvas.width, canvas.height);
// область просмотра - левая нижняя четверть элемента <canvas>
gl.viewport(0, 0, canvas.width/2, canvas.height/2);
// область просмотра - левая верхняя четверть элемента <canvas>
gl.viewport(0, canvas.height/2, canvas.width/2, canvas.height/2);
// область просмотра - правая нижняя четверть элемента <canvas>
gl.viewport(canvas.width/2, 0, canvas.width/2, canvas.height/2);
```

Система координат WebGL

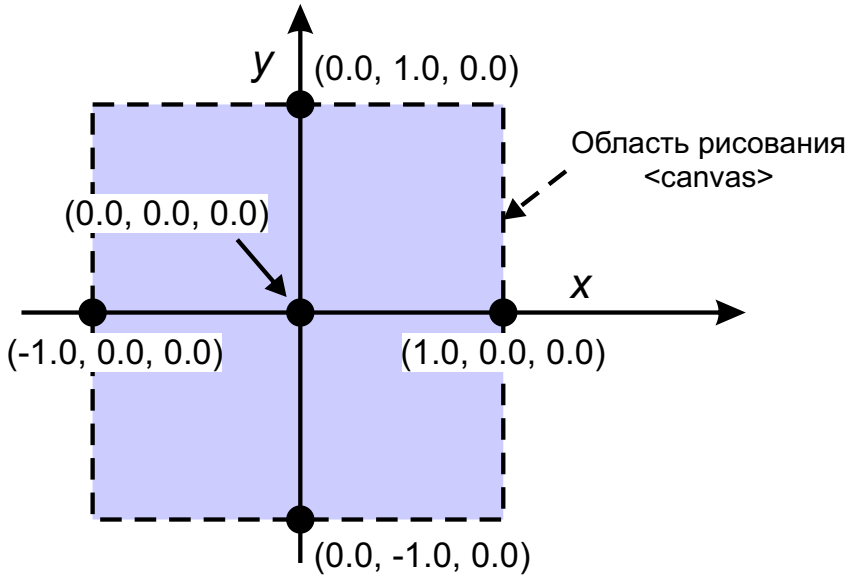






Система WebGL работает с трёхмерной графикой и использует трёхмерную систему координат с тремя осями: X, Y и Z. Ось X простирается по горизонтали, (слева направо), ось Y — по вертикали (снизу вверх) и ось Z — в направлении от плоскости экрана к пользователю. Глаз пользователя находится в начале координат (0.0, 0.0, 0.0) и смотрит вдоль оси Z, в сторону отрицательных значений — за плоскость экрана. Эта система координат является правосторонней и направление её осей можно изобразить пальцами правой руки.

Область просмотра WebGL



Область рисования, которая определяется элементом <canvas>, имеет свою систему координат, отличную от системы координат WebGL, поэтому необходим некоторый механизм отображения одной системы в другую. По умолчанию, WebGL выполняет такое отображение следующим образом:

- центр области рисования <canvas>: $(0.0, 0.0, 0.0)$;
- две вертикальные границы области рисования <canvas>: $(-1.0, 0.0, 0.0)$ и $(1.0, 0.0, 0.0)$;
- две горизонтальные границы области рисования <canvas>: $(0.0, -1.0, 0.0)$ и $(0.0, 1.0, 0.0)$.

Если при рисовании указать координаты вне области просмотра, например $(1.0, 2.0, 0.0)$, рисунок будет обрезан.

Типизированные массивы (typed arrays)

В WebGL часто приходится иметь дело с большими объёмами данных одного типа, такими как координаты вершин и цвета, необходимых для рисования трёхмерных объектов, для чего стандартные JavaScript-числа не очень подходят. Проблемой стандартных массивов является низкая скорость работы и большое потребление памяти. С целью оптимизации для каждого типа данных была добавлена поддержка массивов, элементами которых являются значения конкретного типа (типизированных массивов). Так как тип данных элементов в типизированных массивах известен заранее, такие массивы могут обрабатываться намного эффективнее.

Типизированный массив	Размер в байтах	Описание (тип в языке C)
`Int8Array`	1	8-битовое целое со знаком (`signed char`)
`Uint8Array`	1	8-битовое целое без знака (`unsigned char`)
`Int16Array`	2	16-битовое целое со знаком (`signed short`)
`Uint16Array`	2	16-битовое целое без знака (`unsigned short`)
`Int32Array`	4	32-битовое целое со знаком (`signed int`)
`Uint32Array`	4	32-битовое целое без знака (`unsigned int`)
`Float32Array`	4	32-битовое с плавающей точкой (`float`)
`Float64Array`	8	64-битовое с плавающей точкой (`double`)

Типизированные массивы

Типизированные массивы, как и стандартные, создаются с помощью оператора `new`, которому передаются исходные данные для массива:

```
const vertices = new Float32Array([0.0, 0.5, -0.5, -0.5, 0.5, -0.5]);
console.log(vertices[0]); // 0.0
console.log(vertices[1]); // 0.5
```

Типизированные массивы имеют следующие свойства и константы:

`length`	Число элементов в массиве.
`BYTES_PER_ELEMENT`	Размер одного элемента массива в байтах.

Нельзя присвоить значение элементу типизированного массива с несуществующим числовым индексом, как это возможно с обычными массивами, потому что типизированные массивы проигнорируют такую операцию:

```
vertices[6] = 1.0;
console.log(vertices.length); // 6
console.log(vertices[6]); // undefined
```

Также имеется возможность создать пустой типизированный массив, передав конструктору число элементов в массиве. Например:

```
const vertices = new Float32Array(4);
```

Типизированные массивы

Метод `set()` копирует содержимое другого массива в существующий типизированный массив. Метод `set()` принимает массив (типизированный или обычный) и необязательное смещение, определяющее, откуда начинать вставку данных (значение по умолчанию равно 0). Метод гарантирует, что будут записаны данные только допустимых типов.

```
let ints = new Int16Array(4);
ints.set([25, 50]);
ints.set([75, 100], 2);
console.log(ints.toString()); // 25,50,75,100
```

Метод `subarray()` извлекает часть существующего массива в новый типизированный массив. Метод принимает необязательные начальный и конечный индексы (значение из элемента с конечным индексом не копируется) и возвращает новый типизированный массив. Например:

```
let ints = new Int16Array([25, 50, 75, 100]),
    subints1 = ints.subarray(),
    subints2 = ints.subarray(2),
    subints3 = ints.subarray(1, 3);
console.log(subints1.toString()); // 25,50,75,100
console.log(subints2.toString()); // 75,100
console.log(subints3.toString()); // 50,75
```

Типизированные массивы

Типизированные массивы также включают большое количество методов, функционально эквивалентных одноимённым методам обычных массивов:

`copyWithin()`	`findIndex()`	`lastIndexOf()`	`slice()`
`entries()`	`forEach()`	`map()`	`some()`
`fill()`	`indexOf()`	`reduce()`	`sort()`
`filter()`	`join()`	`reduceRight()`	`values()`
`find()`	`keys()`	`reverse()`	

Типизированные массивы также выполняют проверки, чтобы гарантировать использование данных только допустимого типа. Любое недопустимое значение замещается нулём. Например:

```
let ints = new Int16Array(["hi"]);
console.log(ints.length); // 1
console.log(ints[0]); // 0
```

Поскольку строка не является 16-разрядным целым числом, вместо неё в массив будет записан 0, а свойство `length` массива получит значение 1. Благодаря такой коррекции ошибок методы типизированных массивов не выдают ошибок о присутствии недопустимых данных, потому что недопустимые данные никогда не попадают в массивы.

Типизированные массивы

Несмотря на наличие у типизированных массивов большого количества тех же методов, что и у обычных массивов, у них отсутствуют некоторые типичные методы. Например, следующие методы не поддерживаются типизированными массивами:

`concat()`	`shift()`	`pop()`
`splice()`	`push()`	`unshift()`

Кроме метода `concat()`, все остальные методы могут изменять размер массива. Типизированные массивы имеют фиксированный размер, именно поэтому данные методы недоступны для типизированных массивов. Метод `concat()` не поддерживается, потому что результат конкатенации двух типизированных массивов (особенно если они хранят данные разных типов) выглядит сомнительным, и такая операция, прежде всего, противоречила бы целям применения типизированных массивов.

Повторяющиеся вызовы функции

Обычно, если требуется повторно выполнить некоторую функцию, используются методы `setInterval()` или `setTimeout()`. Однако, поскольку эти методы появились задолго до того, как браузеры стали поддерживать интерфейс с вкладками, они выполняются независимо от того, какая вкладка в данный момент активна. Это может приводить к ухудшению общей производительности, поэтому появился новый метод `requestAnimationFrame()`. Вызов функции, запланированный с помощью этого метода, производится только когда вкладка, где была определена эта функция, активна. Так как метод является новым, он ещё не стандартизован и определен в сторонней библиотеке `webgl-utils.js`, скрывающей различия между разными браузерами.

`requestAnimationFrame(func)`		
Требует от браузера вызвать функцию `func` для перерисовывания. Это требование необходимо повторять после каждого вызова.		
Параметры	`func`	Определяет функцию для вызова.
Возвращаемое значение	Числовой идентификатор запроса	

В этой функции отсутствует возможность указать интервал, через которые должна вызываться функция — функция `func` будет вызвана, когда браузер решит, что веб-страницу нужно перерисовать (обычно перерисовка осуществляется 60 раз в секунду).

Повторяющиеся вызовы функции

Функции `func` передаётся один аргумент, который содержит текущее время (в формате UTC) с минимальной точностью в 1 мс.

```
let start, previousTimeStamp;

function step(timestamp) {
    if (start === undefined)
        start = timestamp;
    const elapsed = timestamp - start;

    if (previousTimeStamp !== timestamp) {
        // do something
    }

    previousTimeStamp = timestamp
    requestAnimationFrame(step);
}
```

Повторяющиеся вызовы функции

Отменить требование на повторный вызов можно с помощью метода:

`cancelAnimationFrame(requestID)`		
Отменяет требование на повторный вызов функции, переданное вызовом `requestAnimationFrame()`.		
Параметры	`requestID`	Значение, возвращаемое вызовом `requestAnimationFrame()`
Возвращаемое значение	нет	