УДК 004.925


А.А. Суконщиков, В.В. Круглов
Вологодский государственный университет

ЭФФЕКТИВНАЯ ВИЗУАЛИЗАЦИЯ МНОЖЕСТВЕННЫХ ПОЛУПРОЗРАЧНЫХ ОБЪЕКТОВ В 3D-ГРАФИКЕ РЕАЛЬНОГО ВРЕМЕНИ

В данной статье рассмотрены наиболее проблемные случаи визуализации полупрозрачных объектов, а также описаны алгоритмы и их комбинации, позволяющие эффективно выполнить визуализацию в таких случаях.

3D-графика реального времени, прозрачность без упорядочивания, система частиц, blending state, видеокарта, Direct3D.

Полупрозрачные объекты и эффекты повсеместно встречаются в окружающей действительности, а это значит, что и в трехмерной графике сложно без них обойтись. Более того, эффект прозрачности зачастую помогает добиться большей наглядности при представлении различных моделей. Вместе с этим из-за аппаратно-программной специфики визуализация прозрачных объектов, а в особенности большого их количества с наложением друг на друга, является проблематичной как в плане достижения высокой эффективности, так и в плане получения корректного результата.

На сегодняшний день в трехмерной компьютерной графике в реальном времени для визуализации применяется единый алгоритм графического конвейера (Rendering Pipeline), который устанавливает перечень стадий визуализации, их порядок, типы входных и выходных данных и доступных ресурсов для каждой стадии и т.д.

На программном уровне, в частности на уровне прикладного программирования, имеются специальные графические API (например, Direct3D или OpenGL), позволяющие программировать работу графического конвейера на видеокарте с последующим его вызовом – draw call'ом. На аппаратном же уровне конвейер поддерживается архитектурой видеокарт, спроектированной специально для эффективной визуализации графики.

Поэтому для достижения максимальной эффективности при визуализации нужно стараться, чтобы вычисления производились на видеокарте, а не на центральном процессоре.

Общий принцип работы конвейера следующий: на основе геометрического описания объектов сцены (координаты их вершин в трехмерном пространстве) по результатам обработки готовое двухмерное изображение выводится на экран. Отдельную сложность для такой визуализации в силу ее специфики могут представлять явления или объекты, не обладающие строго определенной формой, поведением и имеющие некоторую степень прозрачности (например, облака, дым, стекла, огонь, взрывы, струи жидкости), которые тем не менее можно встретить практически повсеместно, а значит, для воссоздания огромного количества сцен реального мира просто необходимо иметь возможность эффективной визуализации подобных эффектов.

Таким образом, главной целью будет нахождение метода визуализации эффектов такого типа средствами одной лишь видеокарты, с минимальным количеством вызовов конвейера.

В ходе достижения цели необходимо будет решить задачи, связанные с изучением необходимых в данном случае возможностей современных графических API, с рассмотрением и выбором имеющихся алгоритмов и техник, полезных для решения данной проблемы, а также с объединением и адаптацией суммы определенных техник к практическому применению с использованием одного графического API – Direct3D 11.0.

Система частиц (Particle System)

В трехмерной графике объекты или явления, не имеющие строго предопределенной формы, принято представлять в качестве так называемой системы частиц (Particle System) – коллекции из ограниченного числа атомарных элементов (частиц), имеющих между собой схожее поведение, но с добавлением в него некоторой доли случайности.

Для имитации эффектов типа огня или дождя эти частицы на каждом кадре визуализации нужно перемещать согласно поведению реализуемого эффекта. Перемещение частиц можно рассчитывать различными способами. В простейшем случае, если имитируется движение с постоянным ускорением, можно воспользоваться выражением из физики:

$$a(t) = v'(t) = p''(t);$$

$$p(t) = \int v(t)dt = \int (ta + v_0)dt = \frac{1}{2}t^2a + tv_0 + p_0.$$

Таким образом, зная начальную позицию, скорость и ускорение, можно определить позицию частицы в любой момент времени. Элемент же случайности в движении частиц вносится добавлением к этим начальным величинам случайных значений.

После смещения частиц нужно обработать их соответствующим образом при помощи конвейера визуализации. При визуализации системы частиц отдельные частицы представляют и передают на конвейер в виде отдельных точек (в Direct3D 11 – это топология прими-

тивов «D3D11_PRIMITIVE_TOPOLOGY_POINTLIST»). Так можно легко рассчитывать перемещение частиц, но точки сами по себе не годятся для визуализации. Поэтому на стадии геометрического шейдера будет производиться развертывание точек в прямоугольники, всегда повернутые в сторону камеры (позиции наблюдения сцены). Чтобы правильно выполнить такое развертывание, нужно знать положение системы координат каждого такого прямоугольника относительно мировой системы координат сцены на каждом кадре. Такую систему координат (x, y, z) можно построить с помощью векторных вычислений, зная позицию камеры (E), позицию точки-частицы (C), а также значение вектора, смотрящего вверх (j):

$$z = \frac{E-C}{\|E-C\|};$$
$$x = \frac{j \times z}{\|j \times z\|};$$
$$y = z \times x.$$

После этого полученные прямоугольники попадают на пиксельный шейдер, где на них накладываются текстуры, соответствующие визуализируемому эффекту (например, изображения клубов дыма или языков пламени).

Система частиц, реализуемая средствами видеокарты (GPU-based Particle System)

Самым простым с точки зрения реализации способом обработки системы частиц является подход [1], где на каждом кадре коллекция частиц (в виде точек) считывается в оперативную память (ОП) из динамического буфера вершин, далее на центральном процессоре (ЦП) производится создание, перемещение или удаление частиц исходя из прошедшего времени или других условий. После обновления частицы снова загружаются в буфер вершин и подаются на конвейер визуализации, где производится их развертывание в прямоугольник, наложение текстур и другие необходимые для визуализации действия.

Главным недостатком такого подхода являются потери производительности [1]. Во-первых, при использовании динамического буфера вершин происходит передача данных с видеокарты (ГП) на центральный процессор и обратно, а такой обмен данными является наиболее медленным по сравнению с передачей ЦП-ОП и ГП-VRAM. Во-вторых, для обновления системы частиц используются вычислительные мощности ЦП, хотя эту работу разумнее переложить на видеокарту, архитектура которой гораздо лучше подходит для параллельных вычислений.

Более эффективным будет подход с реализацией обработки системы частиц полностью на видеокарте (GPU-based Particle System) [1], но в таком случае непременно потребуется несколько вызовов конвейера визуализации, выполняющих разные действия. Здесь нужно остановиться, чтобы понять, как по-разному вызывать конвейер. Разные шейдеры (вершинные, пиксельные и т.д.) вместе со вспомогательными переменными записываются в исходном файле эффектов (.fx). Там же содержатся так называемые pass'ы, служащие для объединения наборов шейдеров в рамках одного draw call'а. Таким образом, имея разные pass'ы, можно, обращаясь к ним по именам, вызывать конвейер по-разному.

Тогда основная логика для обработки частиц будет такой:

- установить буфер вершин с частицами с топологией point-list на вход конвейера и выполнить вызов конвейера визуализации (draw call) с pass'ом, реализующим только лишь вычисления, связанные с обновлением частиц в буфере вершин без, непосредственно, визуализации;
- установить обновленный буфер вершин с частицами на вход конвейера и выполнить draw call с другим pass'ом, производящим саму визуализацию обновленных частиц.

С подобным способом реализации связано несколько трудностей, но современные графические API имеют средства [1], позволяющие воплотить такой алгоритм.

Во-первых, при первом вызове конвейера нам необходимо обновить содержимое буфера вершин и вывести его без визуализации, так как и логика обновления системы частиц и логика развертывания точек в прямоугольник может быть реализована только в геометрическом шейдере (а несколько шейдеров одной стадии не может быть в рамках одного вызова конвейера). Для этого, начиная с Direct3D 10, была добавлена возможность stream-out, то есть вывода вершин после стадии геометрического шейдера в новый буфер, назначенный для этой роли. При создании такого буфера у него должен быть установлен bind-flag «D3D11_BIND_STREAM_OUTPUT».

Во-вторых, по умолчанию после stream-out'а вершины, выведенные в буфер, все равно будут передаваться на следующую стадию конвейера. Чтобы этого не было, нужно отключить растеризацию. Производится это путем установки пустого пиксельного шейдера и отключения проверки глубины в depth-stencil state.

В-третьих, после обновления вершин и вывода их в буфер в первом pass'е мы не будем знать, сколько в итоге вершин будет находиться в буфере (если, конечно, не использовать динамический буфер, но это сильно снизит производительность). При этом для вызова конвейера всегда необходимо указывать количество исходных вершин или индексов. К счастью, вместе с вводом stream-out был добавлен специальный вызов конвейера – DrawAuto, позволяющий вызвать конвейер для тех вершин всех, которые до этого были выведены в stream-out буфер. Отслеживание количества вершин в таком (и только в таком) случае перекладывается на Direct3D.

И в-четвертых, в Direct3D один и тот же буфер не может одновременно быть задан и как буфер входных вершин, и как буфер для stream-out. Данную проблему решит попеременное использование двух буферов.

Проблемы с прозрачностью: ROP, blending

После того, как все прямоугольники частиц прошли пиксельный шейдер, они попадают на финальную стадию визуализации – output merger. На данной стадии производятся так называемые растровые операции – Raster Operations Pipeline (ROP) [2]. Здесь, в частности, производится проверка глубины (depth buffering) полученных пикселей с отбрасыванием более «глубоких», перекрытых более близкими пикселями, а также смешивание цветов пикселей (blending).

Суть blending'а состоит в следующем: для каждого пикселя, полученного в результате работы пиксельного шейдера, производится смешивание его цвета с цветом соответствующего пикселя в буфере кадров по формуле:

$$C = C_{src} * F_{src} + C_{dst} * F_{dst},$$

где C – векторы цветов формата RGBA; F – факторы или сомножители; src (source) – текущий визуализируемый пиксель, dst (destination) – пиксель в буфере кадров.

Параметры вычисления по этой формуле задаются в blending state'ах – COM-интерфейсах Direct3D, являющихся одним из видов render state'ов или состояний визуализации.

При визуализации прозрачных объектов при смешивании применяется так называемый alpha-blending [1], он же OVER operator: в качестве F_{src} используется значение альфа-канала визуализируемого пикселя (т.е. степень его непрозрачности), в качестве же F_{dst} – единица минус степень непрозрачности визуализируемого пикселя. При такой визуализации для получения правильного изображения необходимо, чтобы прозрачные объекты рисовались в порядке от самого дальнего к самому ближнему, ведь только тогда исходное значение C_{dst} при смешивании с очередной прозрачной поверхностью будет верным.

В свою очередь, raster operations pipeline принудительно выполняет операции output merger'а для пикселей в том порядке, в котором соответствующая им геометрия была записана в буфер вершин [3]. Таким образом, при визуализации системы частиц, элементы которой представляют собой полупрозрачные объекты (например, клубы дыма), более новые, то есть более поздно добавленные в буфер частицы будут всегда отображаться поверх более старых, что в конечном итоге для многих ракурсов дает некорректное изображение.

Weighted blended order-independent transparency (OIT)

Самое очевидное решение этой проблемы – сортировка частиц и упорядочивание их в буфере в порядке «от дальних – к ближним», но такой подход приведет к увеличению количества вызовов конвейера и необходимости выполнения сортировки на центральном процессоре, что сильно скажется на эффективности.

На сегодняшний день уже существует несколько методов, позволяющих упростить вычисления в такой ситуации, избавившись от сортировки (order-independent transparency, OIT) [4], но наилучшим образом сочетающим эффективность и корректность визуализации является относительно новый метод под названием «weighted, blended order-independent transparency», что можно перевести как «независимая от порядка прозрачность с использованием весовых значений и смешивания».

Основная идея данного метода и ему подобных заключается в том, что, единовременно приняв полупрозрачные объекты, переданные в произвольном порядке, можно, взяв из них некоторые значения и проведя на основе их вычисления, получить результат хоть и не идентичный, но близкий к тому, который мог бы быть получен при обработке отсортированного набора объектов.

В частности, в weighted, blended OIT для сбора нужных значений используются две текстуры. Первая (accumulation) накапливает в своих пикселях непрозрачные составляющие цветов всех визуализируемых неотсортированных частиц, помноженные на «вес» каждой, зависящий от того, насколько далеко находится та или иная частица от зрителя. Таким образом, если в наборе частиц имеется темная частица, которая при наличии сортировки была бы сзади, но из-за ее отсутствия рисуется перед светлой частицей, перекрывая тем самым ее цвет, то это перекрытие будет скомпенсировано значением «веса».

Вторая (revealage) – ее пиксели накапливают степень непрозрачности (alpha) каждой визуализирующейся частицы из несортированного набора частиц. То есть если, например, некие две частицы покрывают один и тот же пиксель, то alpha для этого пикселя будет рассчитываться дважды, при этом во второй раз будет редактироваться значение непрозрачности, полученное в результате вычислений для первой частицы.

Далее, когда эти текстуры заполнены, значения из них используются для итоговой визуализации эффекта, изображаемого системой частиц: значения цветов пикселей из текстуры accumulation берутся, собственно, как цвета эффекта, визуализируемого системой частиц, а значения из текстуры revealage – как значения непрозрачности результирующего эффекта в разных его участках.

Формулы и более подробное описание идеи алгоритма приведены в [5]. Что же касается реализации, то последовательность ее применительно к системе частиц такова:

- произвести визуализацию всех непрозрачных объектов (если есть);
- вызвать pass, осуществляющий обновление системы частиц;
- вызвать 3D-transparency pass, выполняющий аккумулирование прозрачности по формуле в дополнительные текстуры (render target'ы);
- вызвать 2D-composing pass, где происходит объединение содержимого дополнительных render target'ов.

3D-transparency pass, выполняющий аккумулирование и запись в два вспомогательных буфера кадров, принимает прозрачные объекты в любом порядке без сортировки. Соответственно, для него необходимо создать две дополнительные текстуры и установить их перед его вызовом в качестве render target'ов. Для этого вызова должен быть установлен depth state, выполняющий отбрасывание перекрытых более близкими объектами фрагментов, но не перезаписывающий значения буфера глубины.

Помимо него, для записи правильных значений в дополнительные render target'ы должны быть выставлены blending state'ы для этих буферов, как показано в таблице 1 («r» – цвета фрагментов частиц, «a» – степень непрозрачности фрагментов частиц).

Таблица 1

Blending state для 3D-transparency pass в Weighted blended OIT

render target	формат текстуры D3D	начальное значение RGBA	F_{src}	F_{dst}	C_{src}
accum	DXGI_FORMAT_ R16G16B16A16_FLOAT	(0,0,0,0)	BLEND_ ONE	BLEND_ ONE	(r*a, g*a, b*a, a) * w
revealage	DXGI_FORMAT_R8_ UNORM	(1,0,0,0)	BLEND_ ZERO	BLEND_I NV_ SRC_CO LOR	a

Здесь w – весовое значение, полученное на основе глубины конкретного визуализируемого пикселя. Формула его вычисления в пиксельном шейдере имеет примерно такой вид [5]:

```
w = clamp(pow(min(1.0, color.a * 10.0) +
+ 0.01, 3.0) * 1e8 * pow(1.0 - pin.PosH.z * 0.9, 3.0), 1e-2, 3e3).
```

Таблица 2

Blending state для 2D-composing pass в Weighted blended OIT

render target	формат текстуры D3D	F_{src}	F_{dst}	C_{src}
frame buffer	DXGI_FORMAT_ R8G8B8A8_UNORM	BLEND_SRC_ ALPHA	BLEND_INV _ SRC_ALPH A	(accum.rgb / max(accum.a, 0.0001), 1 - revealage)

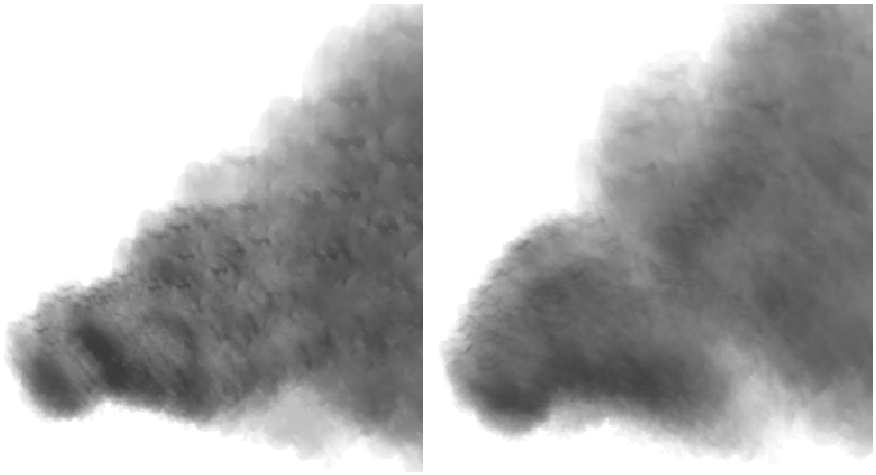

Рис. 1. Сравнение визуализации системы частиц: визуализация с простым alpha blending (слева) и с weighted blended OIT (справа)

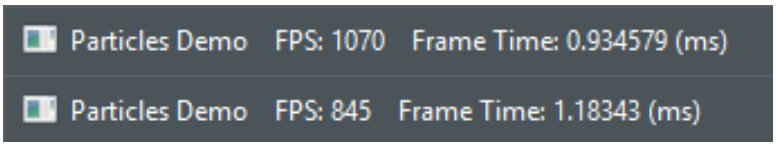

Рис. 2. Сравнение кадровой частоты при визуализации: с простым alpha blending (сверху) и с weighted blended OIT (снизу)

2D composing pass представляет собой визуализацию полноэкранного прямоугольника, на который накладываются пиксели, полученные при совмещении цветов пикселей из двух ранее заполненных текстур, служивших render target'ами. После производится смешивание записанных в этот полноэкранный прямоугольник цветов с текущим содержимым буфера кадров. Параметры blending state'а приведены в таблице 2.

На рисунках 1, 2 приведен результат визуализации системы полупрозрачных частиц, имитирующих дым с освещением на основе карт нормалей сначала с простым alpha blending'ом, затем – с использованием метода weighted blended OIT. При сравнении отчетливо видно, что в первом случае из-за особенностей механизма ROP более новые частицы помещаются в конец буфера и визуализируются в последнюю очередь, перекрывая все про-

чие частицы и создавая тем самым неправдоподобное изображение.

Weighted blended OIT исправляет эти дефекты и дает результат, близкий к тому, который мог бы быть получен при применении alpha blending над отсортированными частицами. Также стоит отметить, что все вычисления, проводимые в данном методе, выполняются на видеокарте, сочетаются с подходом к реализации системы частиц типа «GPU-based particle system» и при этом требуется всего лишь 1 лишний вызов конвейера и 2 чтения из текстур.

Colored blended order-independent transparency (OIT)

Предыдущий описанный метод, несмотря на все его достоинства, не может в неизменном виде применяться для визуализации абсолютно любых полупрозрачных объектов и эффектов. Конкретно это касается объектов, окрашенных в любой цвет, кроме серого [6]. Например: окрашенные жидкости или цветные стекла.

Чтобы объяснить суть проблемы, нужно раскрыть следующие понятия [6]:

- transmittance | transmission – это значение, описывающее свет, а конкретно – его RGB-составляющие, пробивающиеся сзади через полупрозрачную (возможно, окрашенную) поверхность;
- reflectivity – это RGB-значение света, отраженного непосредственно от внешней, самой передней поверхности.

В случае с визуализацией полупрозрачных объектов значение reflectivity будет отображать цвет непосредственно полупрозрачного тела, а transmittance – цвет фона за полупрозрачным объектом, получаемый при прохождении сквозь него.

Таким образом, недостаток визуализации, где учитывается только reflectivity (как в weighted blended OIT) будет в том, что полупрозрачный объект на любом фоне окрашивается всегда одинаково, так как расчет цвет объекта производится только на основе reflectivity, а значит – без учета фона. При учете же значения transmission в визуализации полупрозрачный объект будет всегда окрашиваться по-разному, в зависимости от того, какого цвета объект находится за ним и, соответственно, какой цвет будет пропускать через себя полупрозрачный объект, а какой – не будет. Таким образом, полупрозрачные объекты будут пропускать или поглощать только определенные цвета из меняющегося фона, что в итоге может сильно отразиться на их конечном виде.

Colored blended OIT – это модификация метода weighted blended OIT, где реализован учет значения transmission при расчете цветов полупрозрачных объектов. В основе визуализации все также лежит два pass'а – 3D-transparency pass и 2D-composing pass – и все тот же порядок визуализации – сперва все непрозрачные объекты, после – объекты с прозрачностью.

Отличие 3D-transparency pass в том, что здесь для учета transmission будет использоваться текстура с уже отображенными на ней всеми непрозрачными объектами. Для пикселей этой текстуры, на которые попадает полупрозрачный объект, будет производиться корректировка цветов с учетом того, какова степень прозрачности этого объекта и каков цвет самого фона. Blending state'ы и операции с цветами приведены в таблице 3.

2D-composing pass отличается прежде всего тем, что здесь цвета пикселей back buffer'а уже отредактированы для учета transmission, а это значит, что вместо смешивания по принципу alpha-blending, здесь будет достаточно простого сложения цветов эффекта, полученного с помощью системы частиц и цветов back buffer'а. Параметры blending state'а представлены в таблице 4.

Таблица 3

Blending state для 3D-transparency pass в Colored blended OIT

render target	формат текстуры D3D	начальное значение RGBA	F_{src}	F_{dst}	C_{src}
accum	DXGI_FORMAT_ R16G16B16A16_FLOAT	(0,0,0,0)	BLEND_ ONE	BLEND_ ONE	(r*a, g*a, b*a, a) * w
revealage	DXGI_FORMAT_R8_ UNORM	(1,0,0,0)	BLEND_ ZERO	BLEND_I NV_ SRC_CO LOR	a
color	DXGI_FORMAT_ R8G8B8A8_UNORM	-	BLEND_ ZERO	BLEND_I NV_ SRC_CO LOR	a * (1 - t.rgb)

Таблица 4

Blending state для 2D-composing pass в Colored blended OIT

render target	формат текстуры D3D	F_{src}	F_{dst}	C_{src}
frame buffer	DXGI_FORMAT_ R8G8B8A8_UNORM	BLEND_INV_ SRC_ALPHA	BLEND_ ONE	(accum.rgb / max(accum.a, 0.0001), 1 - revealage)

Другие случаи применения алгоритмов OIT; hardware instancing

Помимо систем частиц, другим частым случаем, в котором могут найти применение алгоритмы по типу colored blended OIT, является визуализация большого числа одинаковых полупрозрачных трехмерных объектов на сцене (например, осколки стекла, стеклянная посуда и тому подобное). Для такой ситуации необходимо упомянуть о технике инстанцирования (instancing), а также о возможностях современных графических API в ее аппаратной реализации. Изначальный смысл инстанцирования заключается в том, что при наличии множества идентичных объектов на сцене, визуализировать их наиболее разумно вместе, не переключаясь на визуализацию других объектов. Так, будет достаточно одной копии повторяющегося объекта в буфере вершин и индексов, а меняться будут только матрицы позиционирования объектов на сцене. Результат – экономия памяти.

Но с большим количеством (одинаковых) объектов связана и другая проблема – для визуализации каждого из них необходим отдельный вызов конвейера визуализации, а это также может привести к нежелательным потерям в эффективности. Для устранения этого недостатка в графических API Direct3D, начиная с Direct3D 10, был введен механизм аппаратного инстанцирования (hardware instancing), при помощи которого можно выполнять визуализацию большого количества одинаковых объектов в одном вызове конвейера. Реализуется данный механизм посредством установки в качестве буфера вершин дополнительного буфера с данными каждого из множества одинаковых объектов (например, с матрицами размещения), а также с помощью специальной команды вызова конвейера для данного случая – «DrawIndexedInstanced». Более подробное описание механизма hardware instancing дано в [1].

Таким образом, в результате исследования был составлен обзор случаев, где в визуализации участвуют полупрозрачные объекты; найдены методы (GPU-based particle system, weighted blended OIT, colored blended OIT, hardware instancing), позволяющие в своем сочетании эффективно реализовать систему полупрозрачных частиц или множество идентичных трехмерных объектов для визуализации различных явлений действительности. Было произведено применение некоторых подходов в тестовой программе с использованием API Direct3D 11.0 и тем самым доказана их применимость и сочетаемость.

Литература

1. Luna F. Introduction to 3D game programming with Directx® 11 – Dulles ; Boston, 2012. – P. 548–556.
2. GPU Gems. Chapter 28. Graphics Pipeline Performance | NVIDIA Developer. – URL : https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-28-graphics-pipeline-performance (дата обращения: 25.10.2021). – Text : Electronic.
3. Rasterizer Order Views 101: a Primer. – URL : https://software.intel.com/content/www/us/en/develop/articles/rasterizer-order-views-101-a-primer.html (дата обращения: 25.10.2021). – Text : Electronic.
4. LearnOpenGL – Weighted Blended. – URL : https://learnopengl.com/Guest-Articles/2020/OIT/Weighted-Blended (дата обращения: 25.10.2021). – Text : Electronic.
5. Casual Effects: Implementing Weighted, Blended Order-Independent Transparency. – URL : http://casual-effects.blogspot.com/2015/03/implemented-weighted-blended-order.html(дата обращения: 25.10.2021). – Text : Electronic.
6. Casual Effects: Fast Colored Transparency. – URL : http://casual-effects.blogspot.com/2015/03/colored-blended-order-independent.html (дата обращения: 25.10.2021). – Text : Electronic.

A.A. Sukonshchikov, V.V. Kruglov
Vologda State University

EFFECTIVE RENDERING OF MULTIPLE TRANSPARENT OBJECTS IN REALTIME 3D GRAPHICS

The article deals with the miost difficult cases of rendering of transparent objects. Italso describes the algorithms and their combinations that allows to make rendering effectively in such cases.

Real-time 3D graphics, order-independent transparency (OIT), particle system, blending state, GPU, Direct3D.