Компьютерная графика
Современные технологии компьютерной графики и рендеринга

Постобработка и спецэффекты

Лекция 7

02.04.02 ФИИТ

Разработка мобильных приложений и компьютерных игр

2025-2026

План лекции

1. Теоретические основы постобработки — закономерности, модели, подходы
2. Классификация эффектов — систематизация по функциональности
3. Оптимизация пайплайна — производительность и качество
4. Коллекция эффектов — 20+ эффектов с кодом и применением
5. Комбинации эффектов — кинематографичные связки
6. Заключение — тренды и ресурсы

Что такое постобработка?

Постобработка — это применение фильтров к уже отрендеренному 2D-изображению сцены

Ключевая парадигма
Вместо моделирования сложных оптических явлений в 3D-пространстве, мы имитируем их как 2D-фильтры после рендеринга

Математическая модель

$$I\_out(x, y) = F(I\_in(x, y), \{\text{параметры}\})$$

где F — оператор эффекта, применяемый к каждому пикселю или окрестности.

Классификация эффектов по способу обработки пикселей

Тип	Описание	Примеры
Точечные	Каждый пиксель обрабатывается независимо	Grayscale, Invert, Color Grading
Окрестностные	Используют соседние пиксели	Blur, Edge Detection, Sharpen
Глобальные	Зависят от всего изображения	Histogram Equalization, Tone Mapping

Сложность вычислений

- Точечные: O(1) на пиксель
- Окрестностные: $O(k^2)$ на пиксель (k — размер ядра)
- Глобальные: O(N) на изображение (N — число пикселей)

Общие принципы. Разделимость операторов

Важнейшая оптимизация

Если ядро свертки разделимо, 2D-свертка заменяется двумя 1D-свертками

Математически

$$K\_2D = K\_H \times K\_V \rightarrow I * K\_2D = (I * K\_H) * K\_V$$

Сложность

- Наивная: O(radius²) сэмплов
- Разделимая: O(2 × radius) сэмплов

Какие эффекты разделимы?

Эффект	Разделим?
Gaussian Blur	✅ Да
Box Blur	✅ Да
Motion Blur (гориз/верт)	✅ Да
Sobel Edge Detection	❌ Нет (но можно разложить)
Sharpen	❌ Нет

Общие принципы. Линейность и нелинейность

Линейные эффекты подчиняются принципу суперпозиции

$$F(a \cdot A + b \cdot B) = a \cdot F(A) + b \cdot F(B)$$

Примеры линейных эффектов:

- Размытие (любое)
- Резкость
- Цветовые матричные преобразования

Нелинейные эффекты нарушают суперпозицию

Примеры нелинейных эффектов:

- Пороговая обработка (Threshold)
- Гамма-коррекция
- Bloom (из-за порога яркости)

Практическое значение:

- Линейные эффекты можно менять местами (коммутируют)
- Нелинейные эффекты требуют строгого порядка в пайплайне

Общие принципы. Пространственная и временная когерентность

Пространственная когерентность

- Соседние пиксели обычно имеют близкие значения
- Позволяет оптимизировать окрестностные эффекты (размытие, edge detection)

Временная когерентность

- Кадры в видео меняются плавно
- Позволяет:
  - Использовать предыдущий кадр для аппроксимации (reprojection)
  - Экономить вычисления для статичных областей

Пример: Temporal Anti-Aliasing (TAA)

```
vec4 currentColor = texture2D(u_current, vTexCoord);
vec4 previousColor = texture2D(u_previous, vPreviousCoord);
vec4 result = mix(currentColor, previousColor, 0.2);
```

Общие принципы. Иерархия разрешений

Mipmap-подход
Многие эффекты можно применять на пониженном разрешении

Классический пример — Bloom:
1. Даунсемплинг до 1/2, 1/4, 1/8 разрешения
2. Размытие на низких разрешениях (быстро)
3. Апсемплинг и наложение

Преимущества
- Уменьшение количества пикселей в 4, 16, 64 раза
- Артефакты малозаметны для эффектов свечения
- Идеально для GPU с ограниченной пропускной способностью

Где применяется
- Bloom
- Depth of Field (быстрая версия)
- Motion Blur
- God Rays

Преимущества такого подхода к реализации эффектов

1. Повышение производительности:
   - Даунсемплинг (уменьшение разрешения) значительно снижает количество пикселей, которые нужно обрабатывать, что существенно ускоряет вычисления.
   - Размытие и другие эффекты на пониженном разрешении выполняются быстрее без потери качества.
2. Оптимизация нагрузки на GPU:
   - Работа с меньшим количеством пикселей требует меньше вычислительных ресурсов.
   - Это особенно важно для мобильных устройств и игр с ограниченными ресурсами.
3. Качество размытия:
   - Размытие на пониженном разрешении выглядит более чётким и менее пикселизированным.
   - Наложение размытого изображения на оригинальное в процессе апсемплинга позволяет сохранить детали и избежать потери информации.
4. Экономия памяти:
   - Уменьшение разрешения временно снижает объём используемой видеопамяти.
   - Это особенно актуально для эффектов, требующих большого объёма памяти (напр: Motion Blur или God Rays).
5. Плавность переходов:
   - Апсемплинг позволяет плавно перейти от размытого эффекта к оригинальному изображению, избегая резких границ.
   - Это создаёт более естественный визуальный результат.
6. Масштабируемость:
   - Такой подход легко масштабируется под разные разрешения экрана без потери производительности.
   - Можно использовать один и тот же алгоритм для разных устройств с минимальными изменениями.

Пример реализации Bloom

1. Даунсемплинг изображения до 1/4 разрешения.
2. Применение размытия (Gaussian Blur) к уменьшенному изображению.
3. Апсемплинг размытого изображения обратно до полного разрешения.
4. Наложение результата на исходное изображение с учётом порогов яркости.

```
// Даунсемплинг
vec2 halfRes = vTexCoord * 0.5;
vec4 downsampled = texture(u_image, halfRes);

// Размытие на пониженном разрешении
vec4 blurred = blur(downsampled, 0.05);

// Апсемплинг
vec4 upsampled = upsample(blurred, 4.0);

// Наложение на исходное изображение
gl_FragColor = upsampled + texture(u_image, vTexCoord);
```

Да, даунсемплинг и апсемплинг действительно требуют вычислительных ресурсов — но их затраты существенно меньше, чем обработка полного разрешения на сложных эффектах

Визуализация эффекта Bloom

Сравнение затрат

Вариант 1. Размытие на полном разрешении (1920×1080 = ~2 млн пикселей)	**Вариант 2. Размытие после даунсемплинга до 1/4 (480×270 = ~129 тыс. пикселей)**
Для размытия с радиусом 15 px требуется ~ 225 сэмплов на пиксель.	Даунсемплинг: 1–2 прохода, ~2 млн операций
Общее количество операций: 2 073 600 × 225 = 466 560 000 сэмплов	Размытие с радиусом 15 px: 129 600 × 225 = 29 160 000 операций
Высокая нагрузка на GPU, особенно при 60 FPS	Апсемплинг: 1 проход, ~2 млн операций
	Наложение: 1 проход, ~2 млн операций
Итого операций: ~ 466 560 000	**Итого операций:** ~35 млн

Экономия: ~24 % по операциям + снижение нагрузки на память

Где именно экономится ресурс

1. Пропускная способность памяти:
   - При обработке полного разрешения GPU постоянно читает/пишет большие текстуры.
   - Даунсемплинг уменьшает объём данных в 16 раз (при 1/4), что снижает нагрузку на шину памяти.
2. Кеш попадания:
   - Маленькие текстуры лучше помещаются в кеш GPU.
   - Локальность данных улучшается — чтение соседних пикселей эффективнее.
3. Параллелизм:
   - GPU обрабатывает пиксели параллельно.
   - На маленьких текстурах больше блоков могут работать одновременно без конкуренции за ресурсы.
4. Количество инструкций:
   - Размытие — $O(n^2)$ операция (зависит от радиуса).
   - Уменьшение n (размера текстуры) даёт квадратичное ускорение.

Технические детали. Даунсемплинг

Цель: уменьшить разрешение текстуры, сохранив важные детали.

Методы

- Простое усреднение

```
vec4 downsample = (
  texture(u_image, uv + offset1) +
  texture(u_image, uv + offset2) +
  texture(u_image, uv + offset3) +
  texture(u_image, uv + offset4)
) * 0.25;
```

- Бикубическая фильтрация — более качественная, но дороже.
- Mipmap генерация — встроенная в GPU функция.

Затраты: 1–4 сэмпла на пиксель выходного изображения.

Технические детали. Размытие на низком разрешении

Оптимизации

- Separable convolution: размытие сначала по горизонтали, затем по вертикали.
  - Вместо 225 сэмплов → 15 + 15 = 30 сэмплов.
- Box blur вместо Gaussian — быстрее, визуально похоже.
- Учёт яркости: размывать только пиксели выше порога (для Bloom).

Пример кода (separable blur)

```
// Горизонтальное размытие
for (int i = -radius; i <= radius; i++) {
  sum += texture(u_texture, uv + vec2(float(i) * pixelSize.x, 0.0)).rgb;
}
sum /= float(radius * 2 + 1);

// Вертикальное размытие (аналогично)
```

Технические детали. Апсемплинг и наложение

Апсемплинг

Цель: вернуть размытое изображение к исходному разрешению

Методы:
- Билинейная фильтрация (по умолчанию в texture)
- Lanczos — качественнее, но медленнее
- Адаптивная фильтрация — разные методы для разных областей

Затраты: 1–4 сэмпла на пиксель (обычно 1 с билинейной фильтрацией)

Наложение

Операции:
- Смешивание размытого слоя с оригиналом: final = original + blurred * intensity.
- Маска по яркости (для Bloom): if (brightness > threshold) add glow.

Затраты: минимальный overhead.

Практические рекомендации

Когда этот подход НЕ выгоден

- Очень маленькие текстуры (< 256×256): накладные расходы на переключение буферов могут превысить экономию.
- Простые эффекты: если эффект требует 1–2 сэмпла (например, виньетка), даунсемплинг не нужен.
- Ограниченная память: если GPU уже на пределе, создание дополнительных текстур опасно.
- Критичная точность: в научных визуализациях потеря данных при даунсемплинге недопустима.

Оптимальные уровни даунсемплинга для Bloom:

- 1/2: для слабых эффектов (радиус размытия < 5 px).
- 1/4: баланс качества и производительности (радиус 5–15 px).
- 1/8: для сильного размытия (радиус > 15 px), часто используют пирамиду из 3–4 уровней.

Как реализовать эффективно:

- Создавайте MIP цепочку текстур заранее.
- Используйте FBO (Framebuffer Objects) для рендеринга в текстуры.
- Применяйте separable convolution для размытия.
- Комбинируйте с маскированием: размывайте только яркие области.
- Тестируйте на целевых устройствах — мобильные GPU чувствительнее к bandwidth.

Пример пайплайна Bloom

1. Отрендерить сцену в HDR текстуру.
2. Даунсемплинг до 1/4.
3. Размытие (horizontal + vertical).
4. Апсемплинг до 1/2.
5. Размытие снова (если нужно сильное размытие).
6. Апсемплинг до полного разрешения.
7. Наложение на оригинал с учётом экспозиции.

Вывод

Да, даунсемплинг и апсемплинг требуют ресурсов, но:

- их сложность линейна (O(n)), тогда как размытие — квадратично ($O(n^2)$);
- они оптимизированы на уровне аппаратного обеспечения GPU;
- экономия на этапе размытия многократно компенсирует затраты.

Итог:

Для тяжёлых эффектов (размытие, Bloom, God Rays) пирамидальная обработка (даунсемплинг → эффект → апсемплинг) почти всегда выгоднее прямой обработки полного разрешения.

Для лёгких эффектов (виньетка, цветокоррекция) достаточно одного прохода.

Общие принципы. Сохранение данных (G-Buffer)

Для сложных эффектов нужна геометрическая информация, потерянная при растеризации.
G-Buffer (Geometry Buffer) — набор текстур:

	Содержание	Размер
Color	Диффузный цвет	RGB8
Normal	Нормали в world/view space	RGB16F
Depth	Глубина (Z)	R32F
Specular	Металличность, шероховатость	RG8

Стоимость:

- Память: до 4 текстур × разрешение
- Пропускная способность: MRT рендеринг
- Компромисс: качество vs производительность

Классификация эффектов — общая схема

ЦВЕТОКОРРЕКЦИЯ

- LUT (Look-Up Table) — цветовая таблица
- HDR Tone Mapping — тональная компрессия для HDR-изображений
- Дополнительные: Color Grading (цветокоррекция), Saturation (насыщенность), Contrast (контраст), Exposure (экспозиция), White Balance (баланс белого)

ГЕОМЕТРИЯ

- Искажения (Warp) — геометрические искажения: Distortion (дисторсия), Chromatic Aberration (хроматическая аберрация), Screen Warp
- Размытие (Blur): Gaussian Blur (гауссово размытие), Motion Blur (размытие движения), Depth of Field (глубина резкости)

СВЕТ

- Bloom / Glow — свечение ярких областей
- God Rays (Crepuscular Rays) — сумеречные лучи / объемный свет
- Lens Flare — блики линз
- Дополнительные световые эффекты: Glare — ослепляющие блики, Volumetric Fog/Lighting — объемный туман и освещение, Screen Space Reflections (SSR) — отражения в экранном пространстве, Ambient Occlusion (SSAO, HBAO) — ambient occlusion

Дополнительные категории: Атмосферные (зерно, виньетка, засветка), Текстурные (пикселизация, линии развертки), Детекционные (границы, силуэты)

Категория 1 — Цветокоррекция

Цель: Управление четкостью, имитация оптики камеры

Эффект	Принцип	Параметры	Применение
Grayscale	Взвешенная сумма каналов	Коэффициенты	Ч/Б стилизация
Sepia	Тонирование в коричневые тона	Интенсивность	Ретро, "старое фото"
Invert	1.0 - цвет	-	Негатив, арт-эффекты
Color Grading (LUT)	Таблица соответствия цветов	3D LUT текстура	Кинематографичная стилизация
HDR Tone Mapping	Сжатие динамического диапазона	Экспозиция, метод	Реалистичное отображение
Posterize	Квантование цветов	Уровни	Стилизация под комиксы


Fix the photo
BEFORE
AFTER
SEPIA ACTION FOR PHOTOSHOP
FREE «SEPIA NATURE»

Color Grading (LUT)


Dynamic Tone Mapping OFF
Dynamic Tone Mapping ON

Категория 2 — Размытие и фокус

Цель: Управление четкостью, имитация оптики камеры

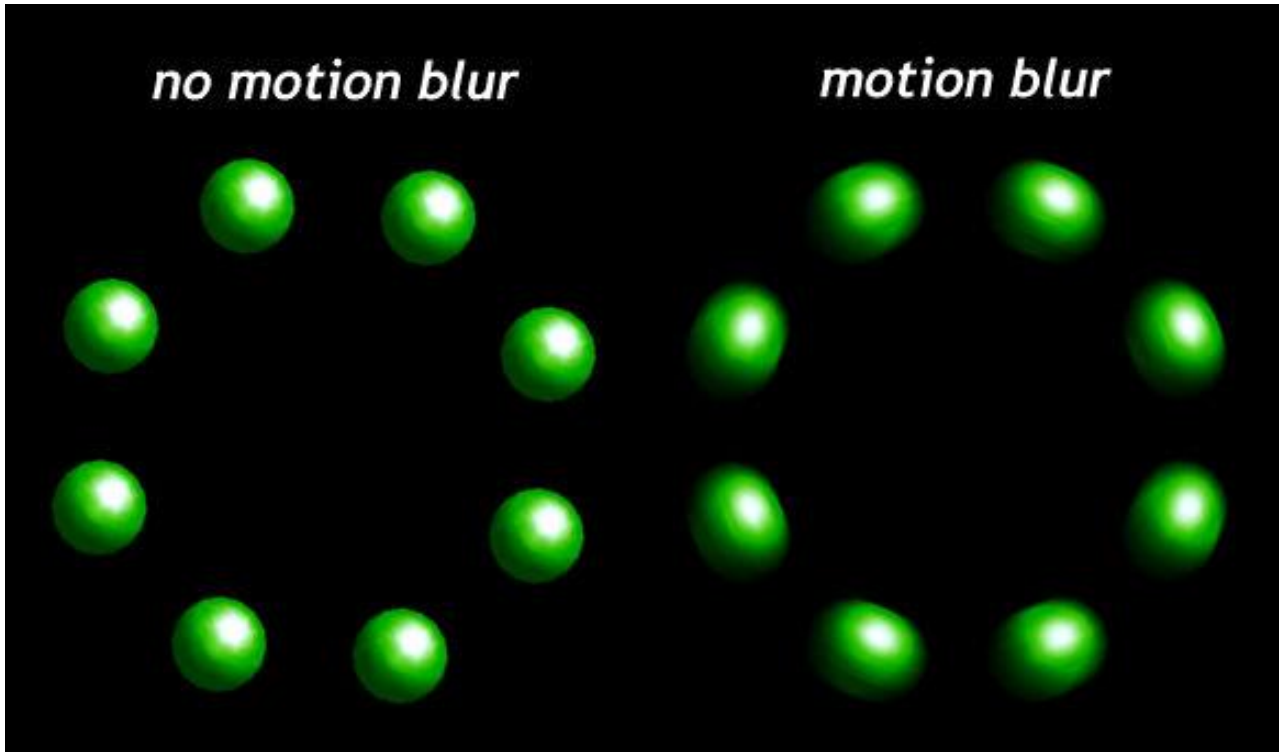
Эффект	Принцип	Сложность	Применение
Gaussian Blur	Свертка с ядром Гаусса	O(radius) с разделением	Универсальное размытие
Box Blur	Среднее арифметическое	O(radius) с разделением	Быстрое размытие
Depth of Field	Размытие по карте глубины	Высокая	Кинематографичный фокус
Motion Blur	Размытие по вектору движения	Высокая	Динамика, скорость
Bokeh	Имитация диафрагмы	Очень высокая	Художественное размытие
Tilt-Shift	Полоса фокуса	Средняя	Эффект "миниатюры"

Motion Blur (Смазывание в движении)

Отсутствие смазывания при движении служит и причиной того, почему движение в играх при 25-30 кадрах в секунду кажется дёрганым, хотя кино и видео при этих же параметрах частоты кадров смотрится прекрасно

Для компенсации отсутствия смазывания в движении желательна или высокая частота кадров (≥60 кадров в секунду) или использование методов дополнительной обработки изображения (motion blur)

Это применяется и для улучшения плавности анимации и для эффекта фото- и кинореалистичности одновременно



Эффект смазывания может быть как полноэкранным (обычно делается постобработкой), так и для отдельных, наиболее быстро движущихся объектов.

Это одна и та же картинка?

Сглаживание переходных процессов во времени

Метод, применяемый для motion blur, называется temporal anti-aliasing, означающий сглаживание переходных процессов во времени

а. Создаем избыточное количество кадров

б. Делим последовательную анимацию на группы по 4 кадра

в. Внутри каждой группы кадров произведём смешивание кадров в один

г. Составим новую анимацию из этих кадров

Имитация motion blur

Разлетающиеся частички раскаленного вещества в месте взрыва можно рендерить в виде коротких линий, а не в виде точек

Это создаст впечатление motion blur

То же самое можно отнести и к процессу визуализации водопадов и выстрелов из оружия

Имитация motion blur

Часто в автосимуляторах можно заметить, что **вращение колес** автомобиля выглядит очень **неестественно**. Колеса автомобиля вращаются с такой бешеной скоростью, что какие-либо детали обода и покрышек просто не должны быть различимы.

Очень удобным решением в этом случае может быть **предварительная подготовка нескольких текстур** колес, просчитанных с различным количеством переходного сглаживания.

И по мере все более и более быстрого вращения колес использовать все более и более размытые текстуры.

Такой же подход возможен и к текстуре дороги.

Примеры motion blur в играх

Depth Of Field (DOF)

Depth of field (глубина резкости), это размывание объектов в зависимости от их положения относительно фокуса камеры.

В компьютерной графике каждый объект отрендеренного изображения идеально четкий, так как линзы и оптика не имитируется при расчетах.

Поэтому, для достижения фото- и кинореалистичности приходится применять специальные алгоритмы.

Эти техники симулируют эффект разного фокуса для объектов, находящихся на разном расстоянии.

Одна из реализаций Depth Of Field

Одним из распространенных методов при рендеринге в реальном времени является смешивание оригинального кадра и его смазанной версии (несколько проходов blur фильтра) на основе данных о глубине для пикселей изображения.

Примеры depth of field в реальном времени

Категория 3 — Искажения (Distortion)

Цель: Нелинейное изменение геометрии изображения

Эффект	Принцип	Математика	Применение
Chromatic Aberration	Смещение RGB-каналов	uv + dir * amount	Ретро, киберпанк
Vignette	Затемнение краев	1.0 - dist * strength	Фокус на центре
Warp / Bulge	Радиальное искажение	uv * (1 + k * r^2)	Рыбий глаз, магия
Twirl	Закручивание вокруг центра	Поворот на angle * r	Психоделика
Kaleidoscope	Симметричное отражение	Полярные координаты	Арт-эффекты
Wave / Ripple	Волновое искажение	uv + sin(uv * freq) * amp	Вода, тепловое марево

Chromatic Aberration

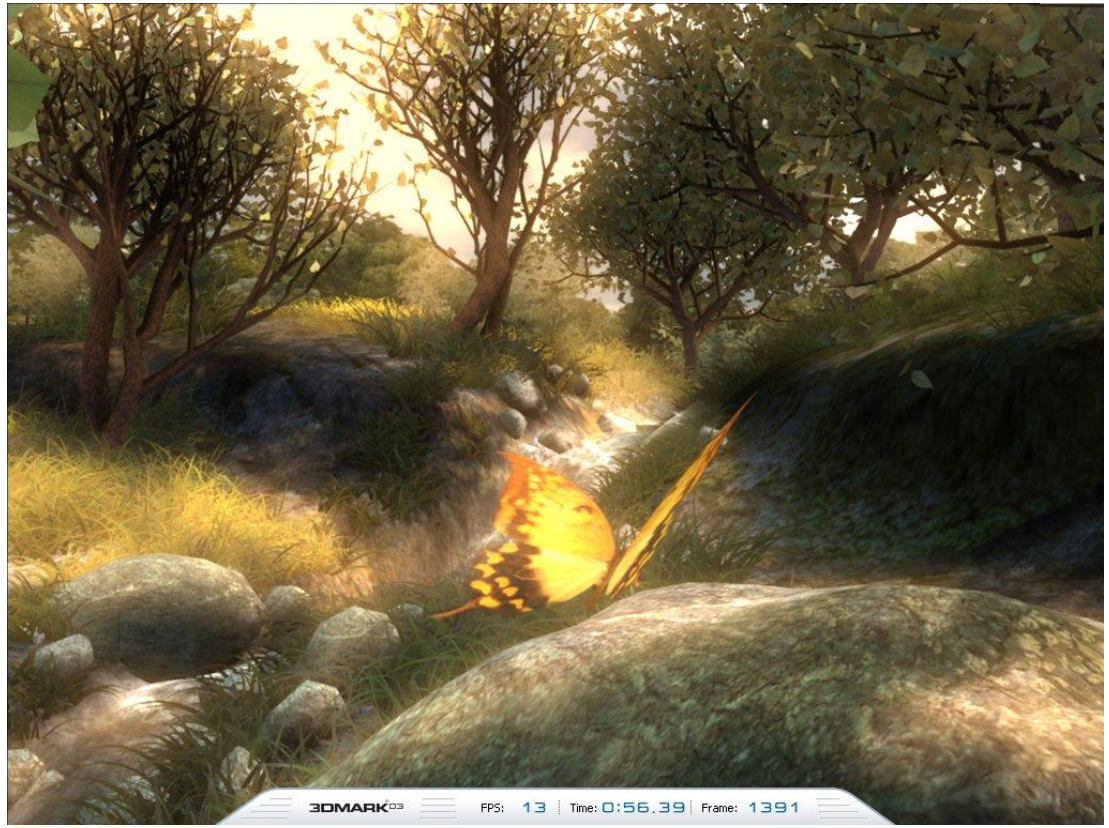
Хроматическая аберрация — искажение изображения, которое возникает из-за зависимости показателя преломления прозрачных сред от длины волны света (дисперсии света) **Проявляется** в том, что световые лучи разных цветов, находящихся на одинаковом расстоянии от оптической оси линзы, преломляются по-разному (имеют разный фокус). Это приводит к **снижению чёткости изображения**, а иногда — к **появлению** на нём **цветных** контуров, полос, пятен — **артефактов**. **Сильнее** всего хроматическая аберрация заметна **по краям** объектов: они **размываются и расслаиваются** на красный, зелёный и синий цветовые каналы. Чаще всего аберрации проявляются **на стыке контрастных границ** — мест, где сталкиваются два разных цвета или яркости

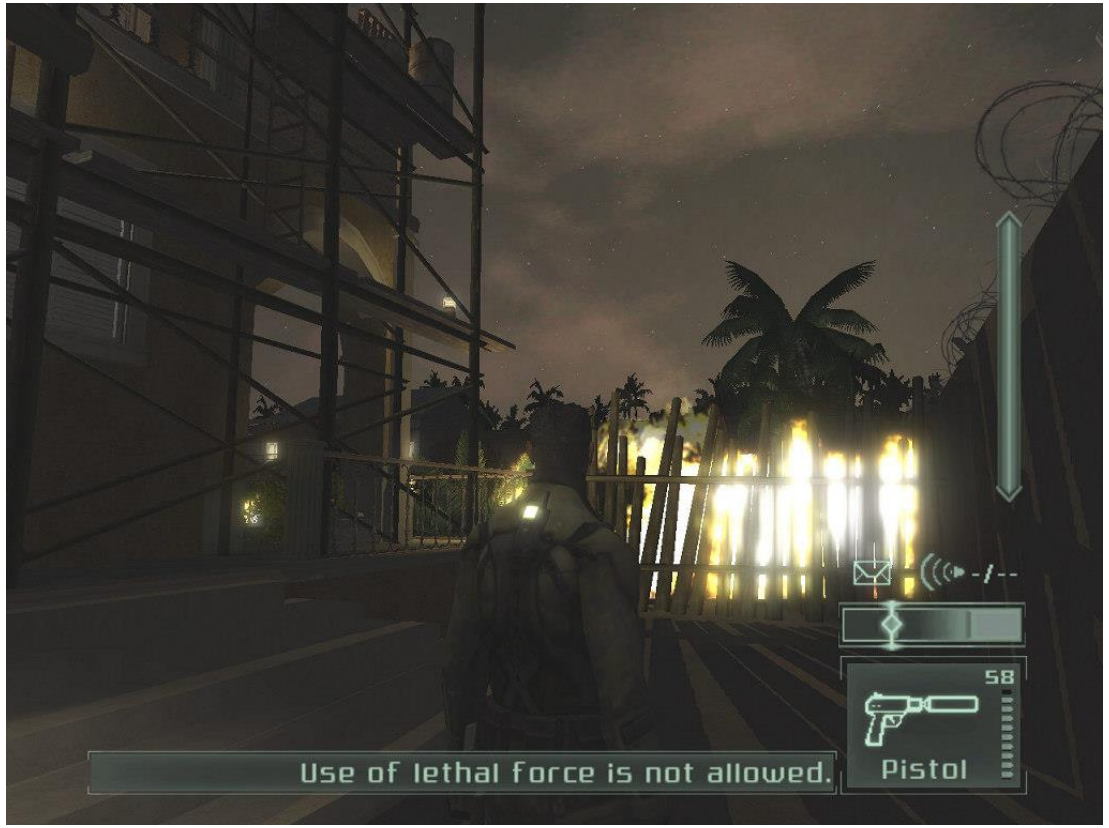
Категория 4 — Световые эффекты

Цель: Добавление источников света, свечения, лучей.

Эффект	Принцип	Проходы	Применение
Bloom	Выделение ярких областей + размытие + наложение	3-5	Свечение, HDR
Glow	Локальное свечение вокруг объектов	2-3	Магия, неон
God Rays	Трассировка лучей от источника	20-40 итераций	Солнечные лучи
Lens Flare	Имитация бликов объектива	Много	Кинематограф
Light Leak	Засветка краев кадра	1	Ретро, аналоговая эстетика

Bloom





Примеры обработки конечного изображения bloom фильтром из 3D приложений реального времени

Как и другие виды постобработки, bloom лучше применять при рендеринге в широком динамическом диапазоне (HDR)

Категория 5 — Текстурные эффекты

Цель: Имитация физических носителей и цифровых артефактов

Эффект	Принцип	Применение
Scanlines	Горизонтальные полосы	ЭЛТ-мониторы, ретро
Pixelate	Увеличение размера пикселя	Цензура, пиксель-арт
Halftone	Точечный растр	Комиксы, газеты
Film Grain	Добавление шума	Кинопленка, атмосфера
Glitch	Смещение каналов, разрывы	Киберпанк, ошибки
Dithering	Структурированный шум	Уменьшение цветов

Категория 6 — Детекционные эффекты

Цель: Выделение структур, границ, объектов

Эффект	Принцип	Применение
Sobel Edge Detection	Градиент яркости	Контуры, cel-shading
Canny Edge Detection	Многоступенчатый	Высокое качество контуров
Silhouette	Обнаружение силуэтов	UI, ассисты
Outline	Обводка объектов	Выделение, игры

Порядок применения эффектов

Типичный пайплайн

1. Цветокоррекция (LUT, Tone Mapping)
2. Геометрические эффекты (искажения, warp)
3. Световые эффекты (Bloom, Glow, God Rays)
4. Размытие и фокус (DOF, Motion Blur)
5. Текстурные эффекты (зерно, линии)
6. Финальные штрихи (виньетка)

Почему такой порядок

- Световые эффекты должны видеть скорректированные цвета
- Размытие после свечения (размывается свет, а не наоборот)
- Виньетка в конце не искажает другие эффекты

Оптимизация. Уменьшение разрешения

Принцип: Тяжелые эффекты применяются к текстуре меньшего размера

Стратегия:

Полное разрешение (1920×1080) → 4.1 млн пикселей
↓
Половина (960×540) → 1.0 млн пикселей (4x меньше)
↓
Четверть (480×270) → 0.25 млн пикселей (16x меньше)

Для каких эффектов работает:
- Bloom (размытие яркой маски)
- Depth of Field (дальний план)
- Motion Blur (приемлемо)
- God Rays (источники света)

Цена: Потеря мелких деталей, но для мягких эффектов незаметно

Оптимизация. Кэширование вычислений

Принцип: Вычисляем один раз, используем много раз.
Пример — предрасчет весов Гаусса:

```
// CPU — javascript
const kernel = [];
for (let i = -radius; i <= radius; i++) {
   kernel.push(exp(-i*i / (2*sigma*sigma)));
}
gl.uniform1fv(u_kernelLoc, kernel);

// GPU — только выборка и умножение — glsl
float weight = u_kernel[i];
sum += texture(u_image, uv + offset) * weight;
```

Где еще применимо:

- LUT (таблица цветов)
- Кубические сплайны для ресемплинга
- Матрицы преобразования цветов

Оптимизация. Ping-Pong FBO

Проблема:
Цепочка эффектов требует множества текстур.

Решение:
Два FBO, чередующихся как чтение и запись.

Схема:
Текстура A (сцена)
↓ Эффект 1
FBO 1 → Текстура B
↓ Эффект 2
FBO 2 → Текстура A
↓ Эффект 3
FBO 1 → Текстура B
↓ Эффект 4
Экран

Примерный код javascript

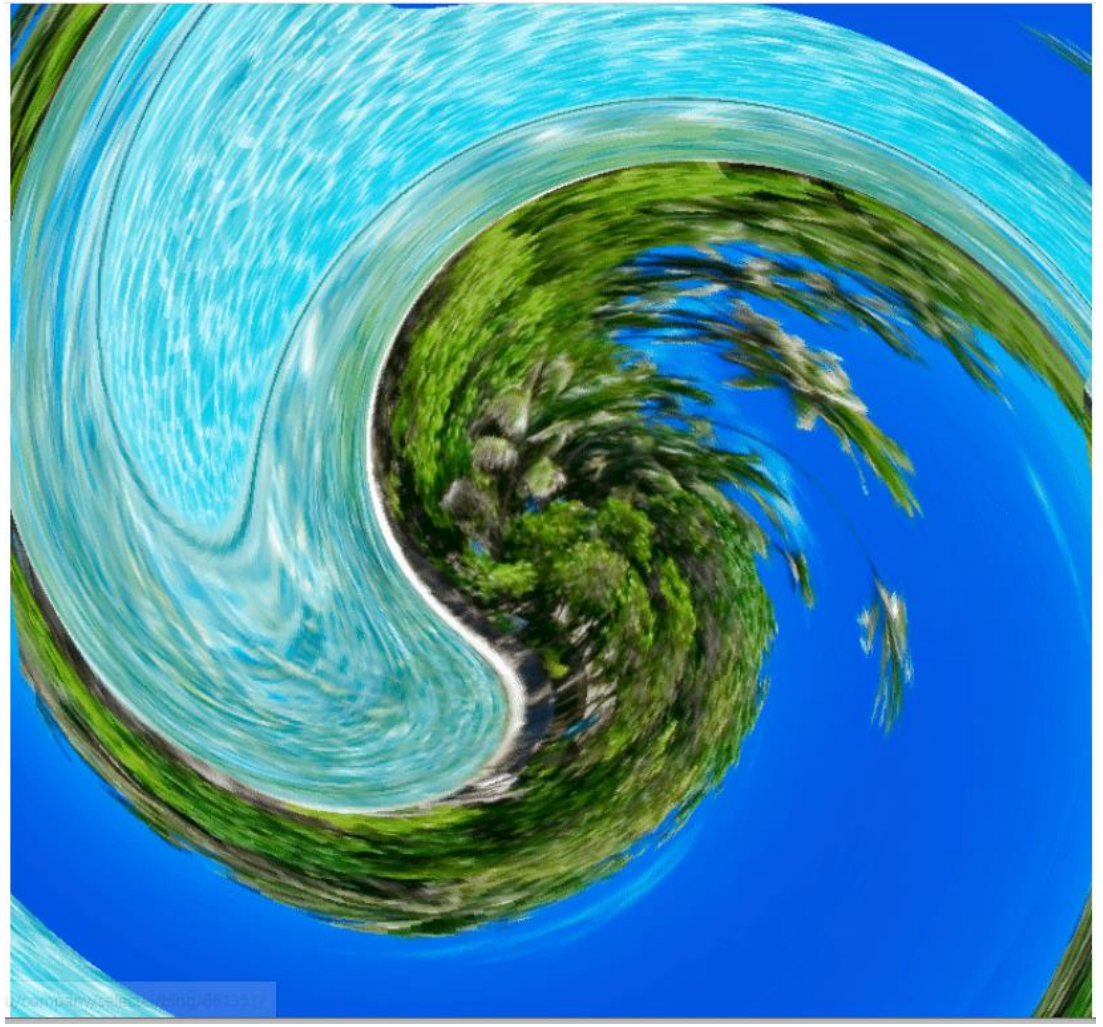
```
let readTexture = sceneTexture;
let writeFBO = fbo1;
let writeTexture = texture1;

for (let effect of effects) {
   gl.bindFramebuffer(gl.FRAMEBUFFER, writeFBO);
   gl.activeTexture(gl.TEXTURE0);
   gl.bindTexture(gl.TEXTURE_2D, readTexture);
   applyEffect(effect);

   // Меняем местами
   [readTexture, writeTexture] = [writeTexture, readTexture];
   [writeFBO, readFBO] = [readFBO, writeFBO];
}
```

```
createFBO() {
    const gl = this.gl;
    const fbo = gl.createFramebuffer();
    // Создаем текстуру с HDR поддержкой и линейным фильтром
    const texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA16F, this.width, this.height, 0, gl.RGBA, gl.FLOAT, null);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
    // Прикрепляем текстуру к FBO
    gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
    // Проверка на завершенность
    if (gl.checkFramebufferStatus(gl.FRAMEBUFFER) !== gl.FRAMEBUFFER_COMPLETE) {
       console.error('FBO incomplete!');
    }
    return { fbo, texture };
 }
```

Какой эффект? Какие идеи реализации?



Fog (Туман)

Окружающий туман — эффект, который используется во всех играх, поскольку позволяет придать объектам или сцене реалистичность, а также почувствовать глубину сцены.

Кроме того, он позволяет скрыть ненужную геометрию.

Интенсивность света в сцене с туманом

В сцене с туманом, интенсивность света, которая достигает камеры пользователя, зависит от трех компонент:

- Поглощение — величина, которая регулирует перекрытие световой интенсивности частиц вдоль трассируемого луча.
- Рассеивание — это выбивание лучей из луча, который идет к глазу.
- Рассеивание внутрь — это попадание (из-за рассеивания) других лучей в луч, который идет к глазу.

Эти три компоненты обобщают и характеризуют одним коэффициентом g, т.е. плотностью

Эффект тумана зависит от трех основных компонент

- Цвет тумана (C)
- Плотность (g)
- Расстояние (d)

Поскольку объекты находящиеся вдали должны больше поддаваться эффекту тумана, ввели коэффициент d, т.е. расстояние.

Окончательный цвет вершины получают интерполированием цвета тумана и текущего цвета

Встроенная реализация GL или через шейдеры

```
glEnable( GL_FOG );
glFogi( GL_FOG_COLOR, color);
Draw();
glDisable( GL_FOG );
```

Линейный туман

Самый простой из всех видов

Основное уравнение:
f = (Zfogend – d)/(Zfogend – Zfogstart)

В таком случае частицы тумана распределяются равномерно от точки Zfogstart и до точки Zfogend.

d – представляет глубину, или расстояние от камеры до точки тумана

f — влияние тумана на цвет вершины, т.е. при 0 — цвет вершины будет цветом тумана (случай когда геометрия находится вдали от зрителя), а при 1 — наоборот соответственно.

Экспоненциальный (квадратичный) туман

Плотность тумана более быстро и плавно уменьшается, чем в случае с линейным туманом, поэтому эффект приближен к реальному эффекту

Основное уравнение для экспоненциального тумана:
f = exp(-d*g)

для экспоненциального квадратичного тумана:
f = exp(-(d*g)*(d*g))

Техника depth fade

В число туманных эффектов входят разные явления, например, пыль и дым.

Обычно такие эффекты представляют собой полупрозрачную плоскость с текстурой и частицами.

Но у такого подхода есть одна проблема — в месте пересечения этой плоскости с геометрией на сцене возникает явная граница, из-за чего эффект выглядит устаревшим и фальшивым.

Тем не менее это самый простой способ его реализации.

Как устроено depth fade?

Сначала идёт визуализация обычной геометрии — туда не входят прозрачные и полупрозрачные объекты.

Это необходимо для заполнения буфера глубины, чтобы знать, как далеко от камеры находится каждый пиксель.

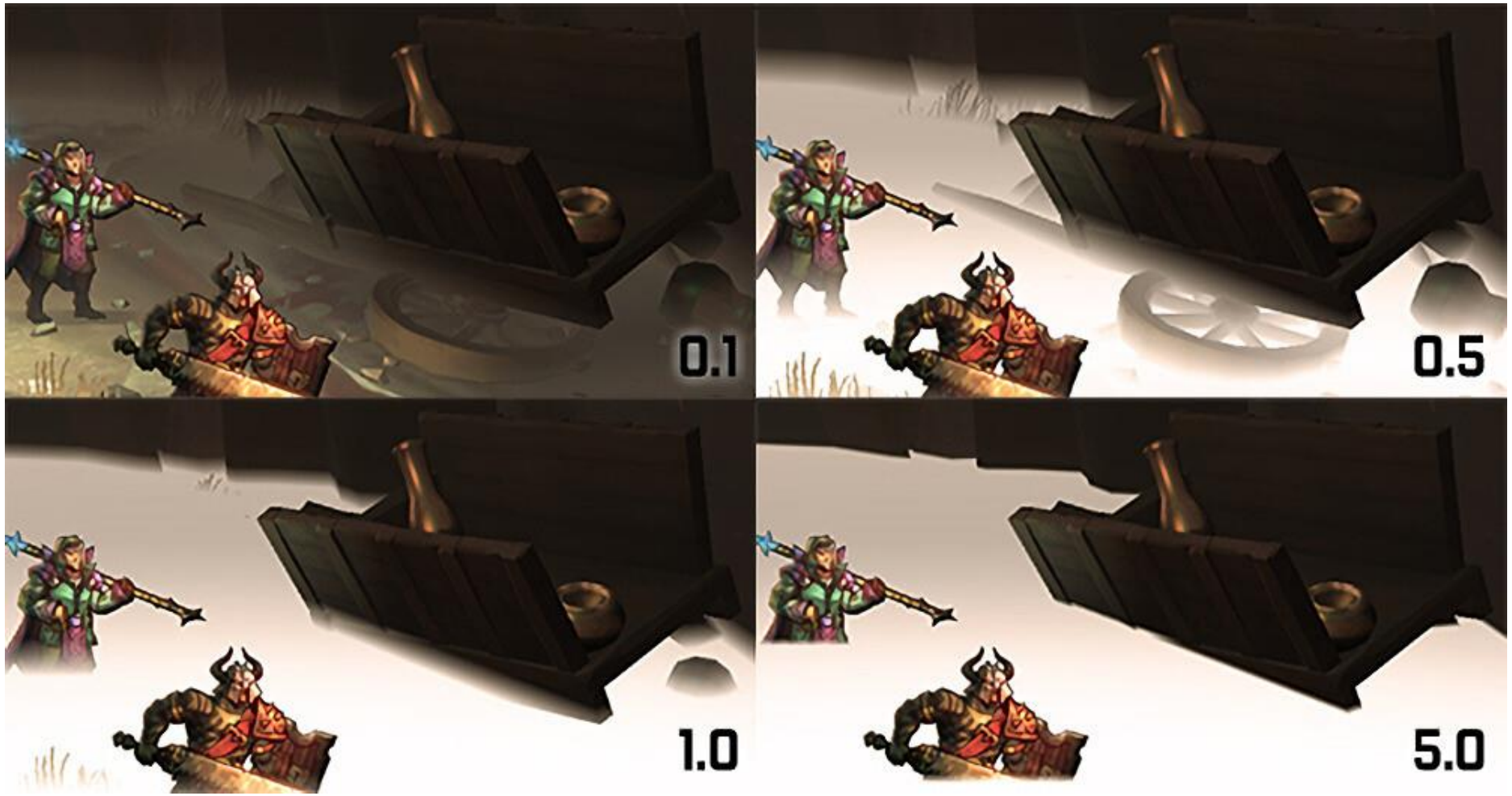
Затем при рендеринге объектов с depth fade фрагментный шейдер оценивает расстояние в буфере глубины и сравнивает его со своими показателями.

Это помогает определить, насколько близко пиксели находятся к области пересечения объекта и плоскости тумана: чем ближе, тем выше прозрачность.

Усиление прозрачности происходит до того момента, пока плоскость тумана не станет полностью невидимой в точке пересечения.

Если изменить расстояние действия эффекта, то можно легко менять плотность тумана.

При высокой плотности тумана эффект сглаживания границ пропадает. Это можно увидеть на нижнем правом примере



Объёмный туман в Red Dead Redemption 2

Объёмные лучи

Программная техника в 3D графике, предназначенная для создания эффекта трёхмерных лучей света, проходящих через окружение трёхмерной сцены.

В реальном мире эффект трёхмерных лучей представлен так называемым оптическим эффектом Тиндаля, когда лучи света проходят через облако газа, дыма, пыли, аэрозоля или пара в затемнённом пространстве.

При прохождении света через такую среду лучи света «подсвечивают» частички пыли или жидкости, которые распылены в воздухе, в результате чего можно увидеть эффект «объёмных лучей».

Ещё одним проявлением подобного оптического эффекта являются сумеречные и противосумеречные лучи.

Фотография. Пример сумеречных лучей, которые возникают во время заката

Фотография. Пример эффекта Тиндаля, когда солнечные лучи проходят через туман

Компьютерная графика: лесная местность в анимационном фильме «Big Buck Bunny», отчётливо видны объёмные лучи.

Эффект прохождения лучей через водную аэрозоль (дождь) Реализация шейдерами

Автор: Акопджанова Николь

Компьютерная графика: объёмные лучи в компьютерной игре «Crysis», которая использует игровой движок «CryEngine 2».

Принцип работы

Объемное освещение требует наличия двух компонентов: **теневой карты** (англ. shadow map) освещаемого пространства и **буфера глубины**.

Для каждого семпла проводится определение: освещается ли он светом из нужного источника света или нет; для этого используются данные из теневой карты. В итоге только освещенные семплы влияют на окончательный цвет пикселя.

Для функционирования в режиме реального времени вышеописанная методика требует оптимизации.

Одним из возможных оптимизационных подходов является рендеринг освещаемого объёма в намного меньшем разрешении чем то, которое используется в изначальном графическом контексте.

Вследствие такого подхода создаются нежелательные эффекты алиасинга, которые могут быть устранены посредством применения фильтра размытия.