Компьютерная графика Современные технологии компьютерной графики и рендеринга

Постобработка и спецэффекты 2

Лекция 8

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 с кодом и применением
5. Комбинации эффектов — кинематографичные связки
6. Заключение — тренды и ресурсы

Эффект 1 — Gaussian Blur (разделяемый)

Принцип: Размытие с весами по распределению Гаусса.
Математика: $G(x,y) = (1/(2\pi\sigma^2)) * e^{(-(x^2+y^2)/(2\sigma^2))}$
Применение: Сглаживание, подготовка к другим эффектам, глубина резкости

Реализация Gaussian Blur — горизонтальный проход

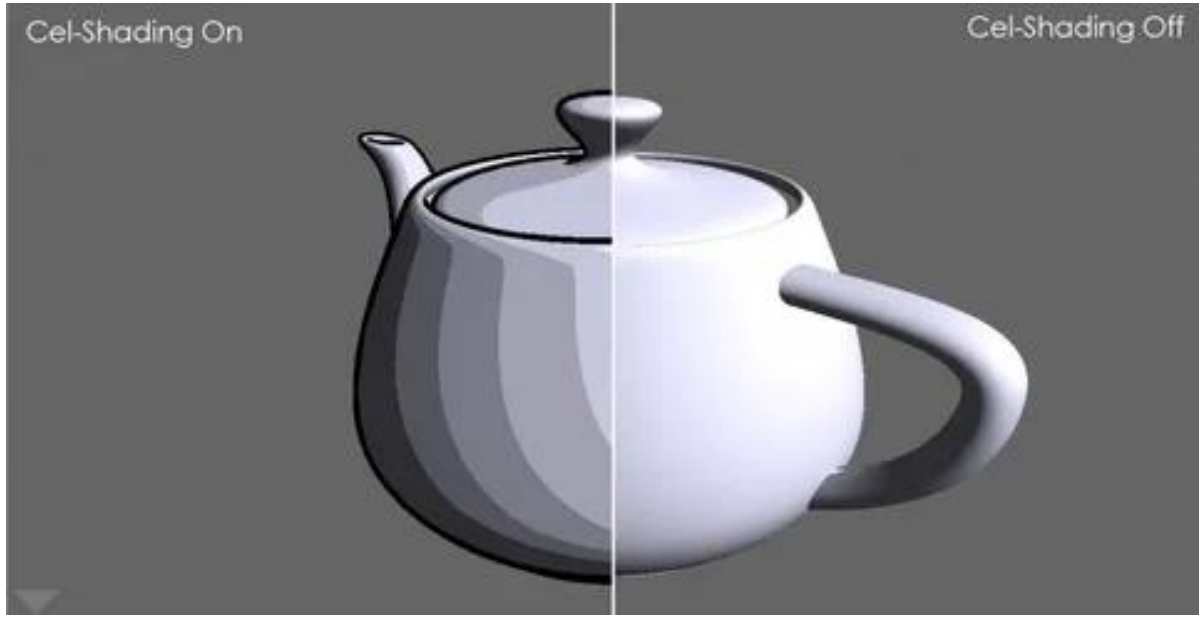
```
uniform sampler2D u_image;
uniform vec2 u_textureSize;
uniform float u_sigma;
varying vec2 vTexCoord;

void main() {
   vec2 onePixel = 1.0 / u_textureSize;
   vec4 sum = vec4(0.0);
   float weightSum = 0.0;

   for (int i = -4; i <= 4; i++) {
      float x = float(i);
      float weight = exp(-(x * x) / (2.0 * u_sigma * u_sigma));
      sum += texture2D(u_image, vTexCoord + vec2(x, 0.0) * onePixel) * weight;
      weightSum += weight;
   }
   gl_FragColor = sum / weightSum;
}
```

Эффект 2 — Sobel Edge Detection

Принцип: Вычисление градиента яркости по X и Y.
Математика:
Gx = [-1,0,1; -2,0,2; -1,0,1] * I
Gy = [-1,-2,-1; 0,0,0; 1,2,1] * I
Edge = $\sqrt{Gx^2 + Gy^2}$
Применение: Контуры для cel-shading, выделение объектов.



Реализация Sobel Edge Detection

```
uniform sampler2D u_image;
uniform vec2 u_textureSize;
varying vec2 vTexCoord;

void main() {
   vec2 offset = 1.0 / u_textureSize;

   // 3x3 блок в оттенках серого
   float tl = texture2D(u_image, vTexCoord + vec2(-1,-1)*offset).r;
   float tc = texture2D(u_image, vTexCoord + vec2( 0,-1)*offset).r;
   float tr = texture2D(u_image, vTexCoord + vec2( 1,-1)*offset).r;
   float ml = texture2D(u_image, vTexCoord + vec2(-1, 0)*offset).r;
   float mr = texture2D(u_image, vTexCoord + vec2( 1, 0)*offset).r;
   float bl = texture2D(u_image, vTexCoord + vec2(-1, 1)*offset).r;
   float bc = texture2D(u_image, vTexCoord + vec2( 0, 1)*offset).r;
   float br = texture2D(u_image, vTexCoord + vec2( 1, 1)*offset).r;

   float gx = -tl - 2.0*ml - bl + tr + 2.0*mr + br;
   float gy = -tl - 2.0*tc - tr + bl + 2.0*bc + br;
   float edge = sqrt(gx*gx + gy*gy);
   gl_FragColor = vec4(vec3(edge), 1.0);
}
```

Эффект 3 — Color Grading с LUT

Принцип: Преобразование цветов по таблице соответствия (3D LUT)
Математика: C_out = LUT(C_in)
Применение:

- Кинематографичная стилизация
- Единая цветовая гамма для игры/фильма
- Быстрая смена настроения сцены

Реализация Color Grading с LUT

```
uniform sampler2D u_image;
uniform sampler3D u_lut;     // 3D текстура 32×32×32
uniform float u_intensity;    // 0-1
varying vec2 vTexCoord;

void main() {
   vec4 color = texture2D(u_image, vTexCoord);
   vec3 lutColor = texture3D(u_lut, color.rgb).rgb;
   vec3 final = mix(color.rgb, lutColor, u_intensity);
   gl_FragColor = vec4(final, color.a);
}
```

Эффект 4 — Bloom (полный пайплайн)

Принцип: Выделение ярких областей, размытие, наложение
Pipeline:

1. Threshold Pass → Bright Pass (яркие пиксели)
2. Downsample (1/2, 1/4, 1/8)
3. Gaussian Blur (на каждом уровне)
4. Upsample + Combine
5. Composite с исходным изображением

Применение: Свечение ярких источников света, HDR-эффекты.

Реализация Bloom (частичная)

Threshold Pass:

```
uniform sampler2D u_image;
uniform float u_threshold;  // 0.8-1.0
varying vec2 vTexCoord;

void main() {
   vec4 color = texture2D(u_image, vTexCoord);
   float brightness = dot(color.rgb, vec3(0.2126, 0.7152, 0.0722));
   if (brightness > u_threshold) { gl_FrargColor = color; }
  else { gl_FragColor = vec4(0.0); }
}

...
```

Composite Pass:

```
uniform sampler2D u_scene;
uniform sampler2D u_bloom;
varying vec2 vTexCoord;

void main() {
   vec4 scene = texture2D(u_scene, vTexCoord);
   vec4 bloom = texture2D(u_bloom, vTexCoord);
   gl_FragColor = scene + bloom;
}
```

Почему именно так, а не просто размыть весь экран?

Производительность: Размыть картинку 1/8 размера ядром 5x5 — дешёво.

Качество: Нельзя сделать огромное свечение (радиус 100 пикселей) простым Гауссом — он будет жутко тормозить. Пирамида даёт большое свечение за O(log n) операций.

Реалистичность: В реальной оптике блики от ярких источников размываются сильнее, чем детали — даунсэмплинг имитирует рассеяние света в линзах.

Эффект 5 — Chromatic Aberration

Принцип: Разделение RGB-каналов с разным смещением.
Математика:

R = I(uv + dir * amount)
G = I(uv)
B = I(uv - dir * amount)

Применение:

- Киберпанк стилизация
- Имитация дешевой оптики
- Эффекты глюка, старой пленки

Реализация Chromatic Aberration

```
uniform sampler2D u_image;
uniform vec2 u_resolution;
uniform float u_amount;  // 0.001 - 0.01
varying vec2 vTexCoord;

void main() {
   vec2 uv = vTexCoord;
   vec2 dir = uv - 0.5;  // направление от центра

   float r = texture2D(u_image, uv + dir * u_amount * 0.3).r;
   float g = texture2D(u_image, uv).g;
   float b = texture2D(u_image, uv - dir * u_amount * 0.3).b;

   gl_FragColor = vec4(r, g, b, 1.0);
}
```

Эффект 6 — Vignette

Принцип: Затемнение краев изображения.
Математика: Vignette = 1.0 - smoothstep(0, radius, distance(uv, center)) * strength
Применение: Портреты, выделение объекта в центре кадра, кинематографичные сцены

Реализация Vignette

```
uniform sampler2D u_image;
uniform float u_strength;  // 0.0-1.0
uniform float u_radius;    // 0.5-1.5
varying vec2 vTexCoord;

void main() {
   vec2 uv = vTexCoord;
   vec2 center = vec2(0.5, 0.5);
   float dist = distance(uv, center);

   float vignette = 1.0 - smoothstep(0.0, u_radius, dist) * u_strength;
   vec4 color = texture2D(u_image, uv);

   gl_FragColor = color * vignette;
}
```

Эффект 7 — Depth of Field (на основе карты глубины)

Принцип: Размытие объектов вне фокуса.
Математика: BlurRadius = abs(Depth - FocusDistance) * Aperture
Применение:

- Фокус на центральном объекте
- Кинематографичный вид
- Создание глубины в композиции

Реализация Depth of Field

```
// Цветное изображение
uniform sampler2D u_image;
// Карта глубины (0 = близко, 1 = далеко)
uniform sampler2D u_depth;
// Разрешение экрана
uniform vec2 u_resolution;
// расстояние фокуса (0-1)
uniform float u_focus;
// сила размытия
uniform float u_aperture;

varying vec2 vTexCoord;

float getBlurRadius(float depth) {
   return abs(depth - u_focus) * u_aperture;
}
```

```
void main() {
   float depth = texture2D(u_depth, vTexCoord).r;
   float radius = getBlurRadius(depth);
   int samples = 1 + int(radius * 16.0);

   vec4 color = vec4(0.0);
   for (int i = 0; i < 16; i++) {
      if (i >= samples) break;
      float angle = float(i) * 0.398;
      float r = sqrt(float(i) / float(samples)) * radius;
      vec2 offset = vec2(cos(angle), sin(angle)) * r / u_resolution;
      color += texture2D(u_image, vTexCoord + offset);
   }
   gl_FragColor = color / float(samples);
}
```

Где используется такой DOF

✅ Плюсы:

- Простота реализации (30 строк кода)
- Работает на любом железе
- Нет артефактов разложения

❌ Минусы:

- Медленный при большом размытии
- Артефакты на границах объектов
- Нереалистичное боке

Применение: Мобильные игры, инди-проекты, прототипы, статические сцены.

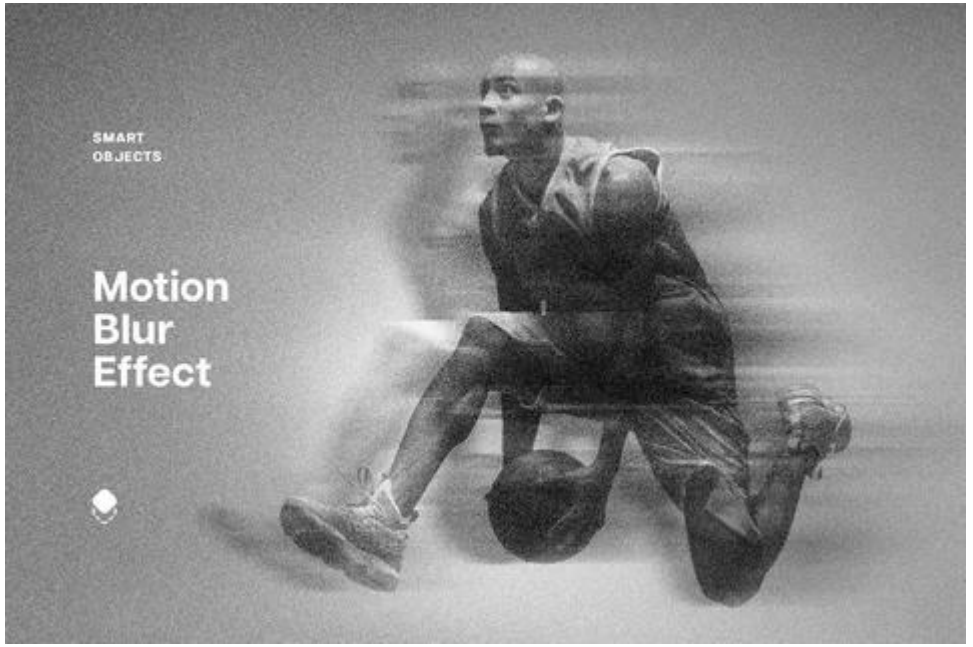
Для продакшена нужны улучшения: даунсэмплинг, сглаживание глубины, правильное боке, но для обучения и прототипов этот код — отличная основа.

Эффект 8 — Motion Blur (на основе velocity buffer)

Принцип: Размытие по направлению движения объекта.
Применение: Динамичные сцены, гонки, экшен, ощущение скорости

Карта скоростей (Velocity Buffer):

- Красный канал (R) = скорость по X (в пикселях за кадр)
- Зеленый канал (G) = скорость по Y (в пикселях за кадр)
- Диапазон: обычно от -1000 до +1000 пикселей



Реализация Motion Blur (на основе velocity buffer)

```
uniform sampler2D u_image;     // Цветное изображение (текущий кадр)
uniform sampler2D u_velocity;  // карта скоростей (RG = вектор)
uniform vec2 u_resolution;     // Разрешение экрана
uniform int u_samples;         // Количество сэмплов (8-16)
varying vec2 vTexCoord;

void main() {
   vec2 velocity = texture2D(u_velocity, vTexCoord).rg;
   vec4 color = vec4(0.0);

   for (int i = 0; i < u_samples; i++) {
      // t — время от 0 до 1, равномерно распределённое между сэмплами
      float t = float(i) / float(u_samples - 1);
      vec2 offset = velocity * (t - 0.5); //  центрирует сэмплы относительно текущего кадра
      //  центрирует сэмплы относительно текущего кадра
      color += texture2D(u_image, vTexCoord + offset / u_resolution);
   }
   gl_FragColor = color / float(u_samples);
}
```

Эффект 9 — God Rays (лучи света)

Принцип: Трассировка лучей от источника света с затуханием
Идея: Для каждого пикселя нужно протянуть линию к источнику света (лампе, солнцу) и собрать яркость всех пикселей на этой линии.
Применение: Солнечные лучи, атмосферные эффекты, магия

Реализация God Rays (лучи света)

```
//Сцена (обычное изображение)
uniform sampler2D u_scene;
//Маска ярких областей
//(обычно пороговая версия сцены)
uniform sampler2D u_lightMask;
// позиция источника (UV)
uniform vec2 u_lightPos;
// Затухание (каждый следующий сэмпл слабее)
uniform float u_decay;    // 0.9-0.98
// Количество шагов (качество лучей)
uniform int u_iterations; // 20-40
varying vec2 vTexCoord;
```

```
void main() {
   vec2 uv = vTexCoord;
   vec2 delta = uv - u_lightPos;
   delta /= float(u_iterations);

   vec4 color = texture2D(u_scene, uv);
   vec4 glow = vec4(0.0);

   for (int i = 0; i < u_iterations; i++) {
      uv -= delta;
      glow += texture2D(u_lightMask, uv) * u_decay;
   }

   glow *= 0.2;
   gl_FragColor = color + glow;
}
```

Где применяется этот эффект

Игра/Сцена	Применение
Skyrim	Солнечные лучи сквозь облака
God of War	Лучи сквозь листву деревьев
Dark Souls	Свет из окон в тёмных комнатах
Подводные сцены	Лучи солнца сквозь воду

Параметры настройки

Параметр	**Эффект**	**Типичные значения**
u_iterations	Длина лучей (больше = длиннее)	16-64
u_decay	Затухание (меньше = короче лучи)	0.8-0.95
u_lightPos	Позиция источника	Зависит от сцены
glow *= 0.2	Яркость лучей	0.1-0.5

Эффект 10 — Pixelate

Принцип: Увеличение размера пикселя (группировка).
Применение: Цензура, ретро-стилизация, пиксель-арт эстетика.

Реализация Pixelate

```
uniform sampler2D u_image;
uniform vec2 u_resolution;
uniform float u_pixelSize;  // 2.0 - 20.0
varying vec2 vTexCoord;

void main() {
   vec2 pixel = u_resolution / u_pixelSize;
   vec2 uv = floor(vTexCoord * pixel) / pixel;
   gl_FragColor = texture2D(u_image, uv);
}
```

Что происходит:

Превращаем непрерывные UV-координаты в дискретные (ступенчатые)

Все пиксели в одном блоке получают одинаковые UV

Сэмплируем текстуру по этим UV → весь блок одного цвета

Вариации эффекта Pixelate

1. Прямоугольные пиксели (не квадратные)
2. Сглаженная пикселизация (с усреднением)
3. Динамическая пикселизация (эффект старой игры)
4. Пикселизация с маской (только для части экрана)

Где применяется

Применение	**Пример**
Retro-игры	Minecraft (режим пикселизация), Hotline Miami
Сны/флэшбеки	В кинематографе для передачи воспоминаний
Цензура	Размытие лиц или номеров машин (пикселизация вместо мозаики)
Лоу-поли стиль	Намеренное снижение детализации
UI-эффекты	Анимированные пиксельные переходы

Эффект 11 — Film Grain

Принцип: Добавление шума для имитации пленки.
Идея: В реальной плёнке или при съёмке с высоким ISO сенсор камеры создаёт случайные шумы — яркие и тёмные точки. Этот эффект **добавляет псевдослучайный шум** к каждому пикселю.
Применение: Кинопленка, атмосфера, борьба с бэндингом.

Реализация Film Grain

```
uniform sampler2D u_image;
uniform float u_intensity;  // 0.05-0.2
uniform float u_time;
varying vec2 vTexCoord;

float random(vec2 st) {
   // Это классическая хеш-функция для генерации псевдослучайных чисел в шейдерах
   // (где нет встроенного random).
   return fract(sin(dot(st.xy, vec2(12.9898,78.233))) * 43758.5453123);
}

void main() {
   vec4 color = texture2D(u_image, vTexCoord);
   float grain = random(vTexCoord + u_time) * u_intensity;
   gl_FragColor = color + (grain - 0.5) * u_intensity;
}
```

Разбор функции по шагам:

```
return fract(sin(dot(st.xy, vec2(12.9898,78.233))) * 43758.5453123);


// 1. Берём координаты точки (st.x, st.y) и
vec2(12.9898, 78.233)  // Магические константы (простые числа)

// 2. Скалярное произведение (dot) — смешиваем x и y в одно число
float d = st.x * 12.9898 + st.y * 78.233;

// 3. Синус — создаёт хаотичные колебания
float s = sin(d);

// 4. Умножение на большое число — усиливает хаос
float m = s * 43758.5453123;

// 5. fract() — берём дробную часть (от 0 до 1)
return fract(m);
```

Получение случайного зерна

float grain = random(vTexCoord + u_time) * u_intensity;
Почему vTexCoord + u_time?

vTexCoord — уникально для каждого пикселя (разные координаты → разное случайное число)

+ u_time — меняет входные данные с каждым кадром → зерно мерцает

Параметры

Диапазоны u_intensity:

- 0.05 → слабое зерно (едва заметно)
- 0.1 → среднее зерно (кинематографичный эффект)
- 0.2 → сильное зерно (старая плёнка)

Зачем u_time:

- Чтобы зерно мерцало от кадра к кадру (как реальная плёнка).

Улучшенные версии

1. Цветное зерно (как реальная плёнка)
2. Зависимость от яркости (яркие участки чище)
3. Масштабированное зерно (более натуральное)

Где применяется

Применение	**Пример**
Киноэффект	Кинематографичные игры (The Last of Us, Red Dead Redemption 2)
Ретро-хорроры	Silent Hill, Resident Evil (зерно для атмосферы)
Флэшбеки/воспоминания	Визуальный маркер нереального
Симуляция плёнки	Фото-приложения, Instagram-фильтры
Маскировка артефактов	Скрывает banding (полосатость) в градиентах

КОМБИНАЦИИ ЭФФЕКТОВ

Комбинация 1 — Кинематографичный вид

Состав:

1. Color Grading (LUT — Teal & Orange)
2. Depth of Field (фокус на объекте)
3. Vignette (легкое затемнение)
4. Film Grain (мелкое зерно)

Применение: Кат-сцены, портреты, трейлеры.

Параметры:

- DOF: фокус на персонаже, размытие фона
- Vignette: интенсивность 0.2-0.3
- Grain: 0.05-0.1

Комбинация 2 — Киберпанк / Неон

Состав:

1. Bloom (интенсивное свечение)
2. Chromatic Aberration (RGB split)
3. Scanlines (тонкие линии)
4. Color Grading (пурпурный + голубой)

Применение: Городские сцены, киберпанк-стилистика.
Параметры:

- Bloom: порог 0.7, интенсивность 1.5
- Chromatic: amount 0.005
- Scanlines: толщина 2px, яркость 1.2

Комбинация 3 — Ретро / ЭЛТ

Состав:
1. Pixelate (умеренная пикселизация)
2. Scanlines (ярко выраженные)
3. Film Grain (шум)
4. Posterize (4-6 уровней)

Применение:
Инди-игры, ретро-эстетика

Комбинация 4 — Постапокалипсис / Хоррор

Состав:

1. Chromatic Aberration (сильная)
2. Film Burn (засветки)
3. Vignette (темные края)
4. Grain (крупный шум)

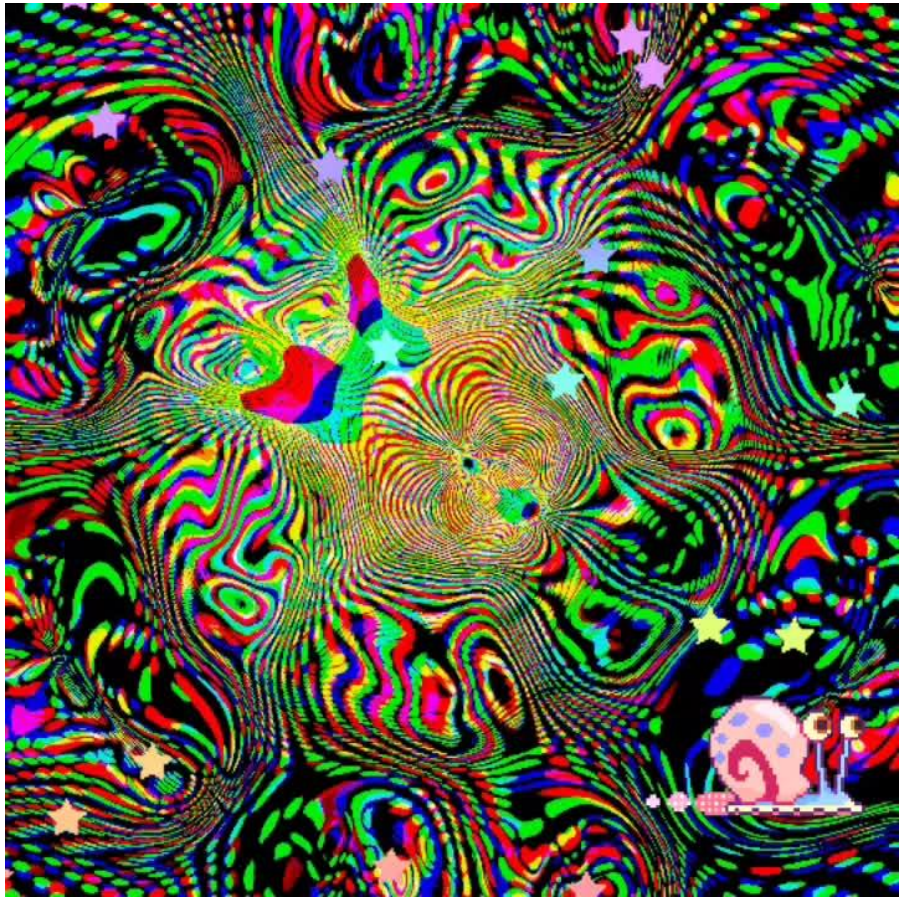
Применение: Хоррор-игры, сцены разрушений

Комбинация 5 — Сюрреализм / Психоделика

Состав:

1. Kaleidoscope
2. Twirl
3. Color Cycle (анимация оттенка)
4. Warp

Применение: Арт-проекты, музыкальные клипы.



Итоги

1. Закономерности постобработки
    - Разделимость операторов
    - Линейность и нелинейность
    - Иерархия разрешений
    - G-Buffer для геометрии
2. Классификацию эффектов
    - Цветокоррекция
    - Размытие и фокус
    - Искажения
    - Световые эффекты
    - Текстурные эффекты
    - Детекционные эффекты
3. Набор эффектов с реализацией
4. Элементы оптимизации пайплайна
5. Комбинации эффектов для разных стилей

Практические рекомендации

Для разработки:

1. Начинайте с простых эффектов (Grayscale, Invert)
2. Используйте Ping-Pong FBO для цепочек
3. Профилируйте производительность
4. Снижайте разрешение для тяжелых эффектов
5. Комбинируйте эффекты в разумных пределах (3-5)

Для выбора эффектов:

- Игры: Bloom, DOF, Motion Blur, Color Grading
- UI/UX: Vignette, Glow на выделенных элементах
- Арт: Kaleidoscope, Glitch, Warp, Halftone
- Ретро: Scanlines, Pixelate, Film Grain

Типичные ошибки и их решение

Ошибка	Решение
Размытие в гамма-пространстве	Перевести в линейное пространство
Слишком много проходов	Объединять эффекты в один шейдер
Артефакты на границах	Проверить фильтрацию текстур
Падение FPS	Снизить разрешение, уменьшить сэмплы
Цвета выглядят плоско	Добавить цветокоррекцию (LUT)

Перспективы развития

1. Реалтайм-композитинг:
    - Смешивание 3D-графики с видео в реальном времени
    - Использование WebRTC + WebGL
2. WebXR:
    - Постобработка для VR/AR
    - Компенсация искажений линз
    - Foveated rendering (снижение качества на периферии)
3. ИИ в постобработке:
    - DLSS (Deep Learning Super Sampling) — апскейлинг с ИИ
    - FSR (FidelityFX Super Resolution) — пространственный апскейлинг
    - Нейросетевые фильтры (стилизация)
4. Вычислительные шейдеры (WebGPU):
    - Более эффективные алгоритмы
    - Параллельная обработка не только пикселей

Сравнение DLSS vs FSR

Характеристика	DLSS (NVIDIA)	FSR (AMD)
Апскейлинг с ИИ	✅ Да (нейросеть)	❌ Нет (алгоритм)
Требования к железу	RTX (тензорные ядра)	Любая видеокарта
Качество при 4K	★★★★★ (отлично)	★★★★☆ (хорошо)
Качество при 1080p	★★★★☆	★★★☆☆ (заметнее)
Прирост FPS	2-3x	1.5-2x
Артефакты	Редко (призраки)	Редко (рябь)
Где работает	Windows, частично Linux	Windows, Linux, PlayStation, Xbox

Сравнение всех трёх технологий

Технология	Цель	Использует ИИ?	Требования	Прирост FPS
DLSS	Апскейлинг + сглаживание	✅ Глубокая нейросеть	NVIDIA RTX	+100-300%
FSR	Апскейлинг + резкость	❌ Математический алгоритм	Любая карта	+50-150%
Нейросетевые фильтры	Изменение стиля	✅ Сверточные сети (CNN)	Любая карта (но медленно)	-30-70% (накладные расходы)

Вопросы для самопроверки

1. В чем разница между точечными и окрестностными эффектами?
2. Какие эффекты разделимы и почему это важно?
3. Почему порядок применения эффектов важен?
4. Как оптимизировать Bloom для мобильных устройств?
5. Что такое G-Buffer и зачем он нужен?
6. Как адаптивно управлять качеством постобработки?
7. В каких случаях нужно применять постобработку в линейном пространстве?