



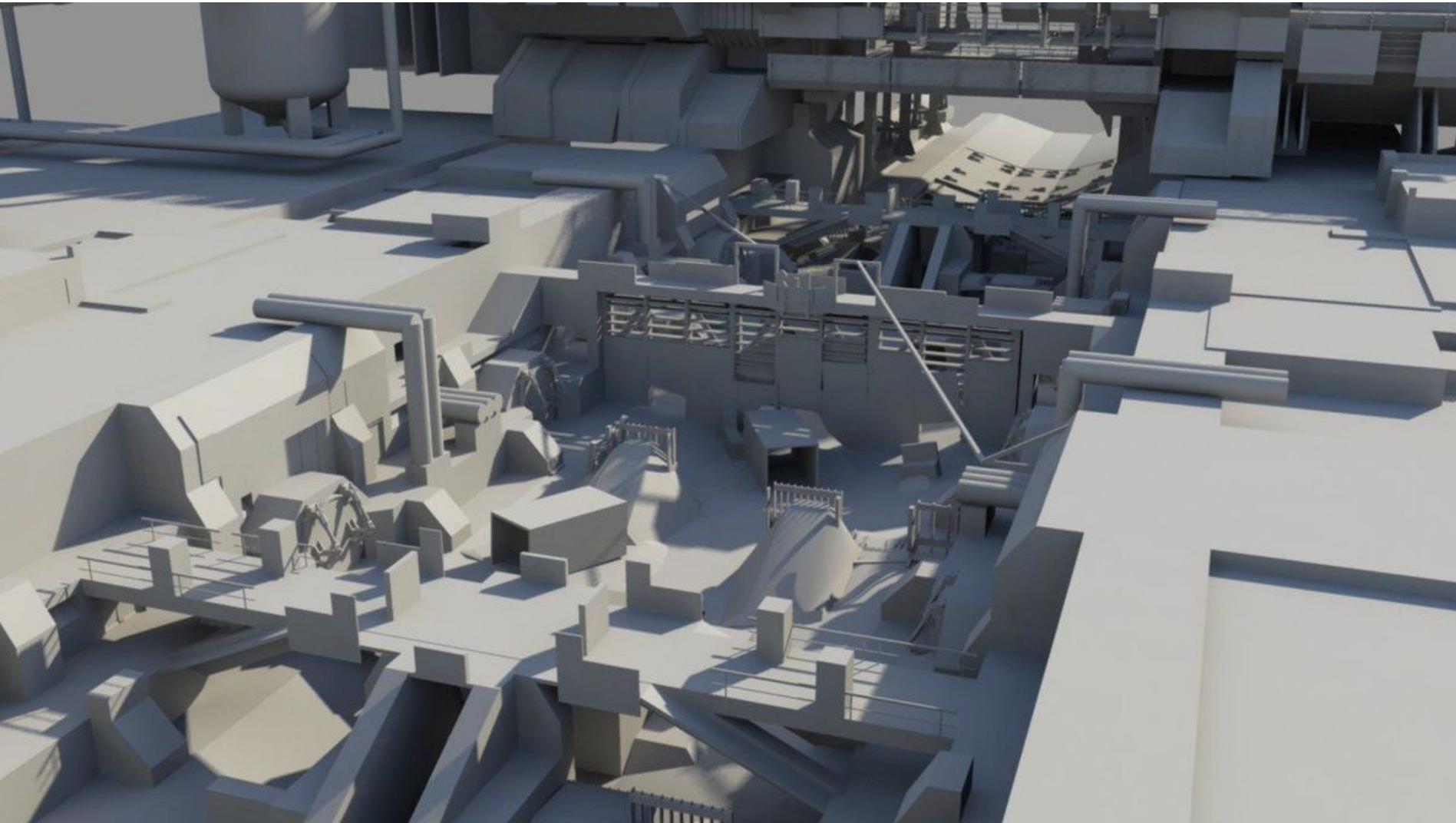
**МИРЭА - РОССИЙСКИЙ
ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ
(www.mirea.ru)**

Лекция: Level design. Подходы в разработке и оптимизации

Лектор: Синицын Анатолий Васильевич

**Кафедра инструментального и прикладного программного
обеспечения**

Что такое дизайн уровней?



Что такое геймплей?



Дизайнер уровней ≠ художник по окружению

Дизайнер уровней



Идея



Как сделать уровень запоминающимся? Изюминка



Достопримечательности



Геймплейная находка. Выразительный
образ



Три в одном флаконе



Вау-моменты. Катастрофы



Гипертрофированный масштаб



Впечатляющая панорама



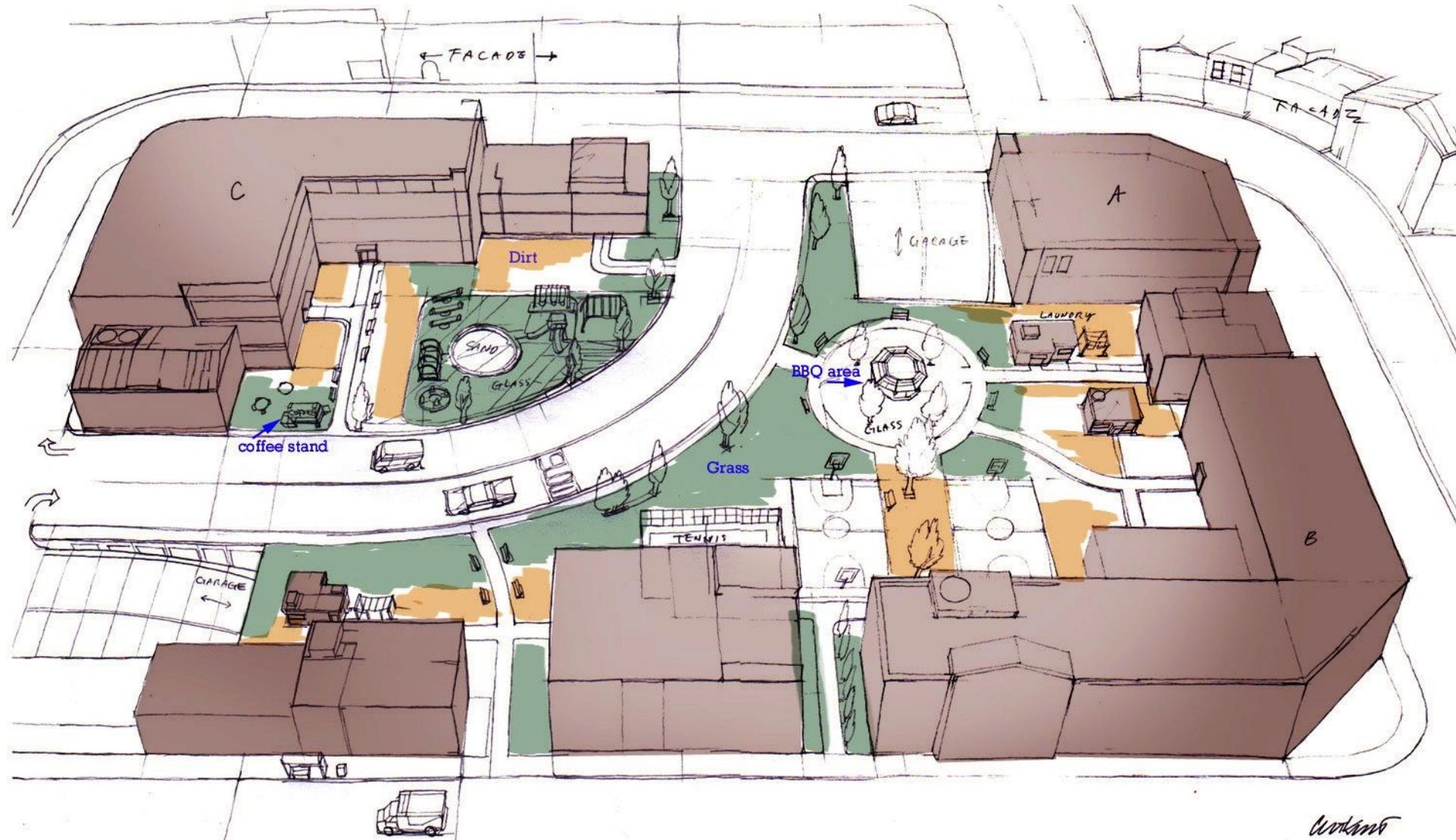
Шокирующие сцены жестокости и насилия



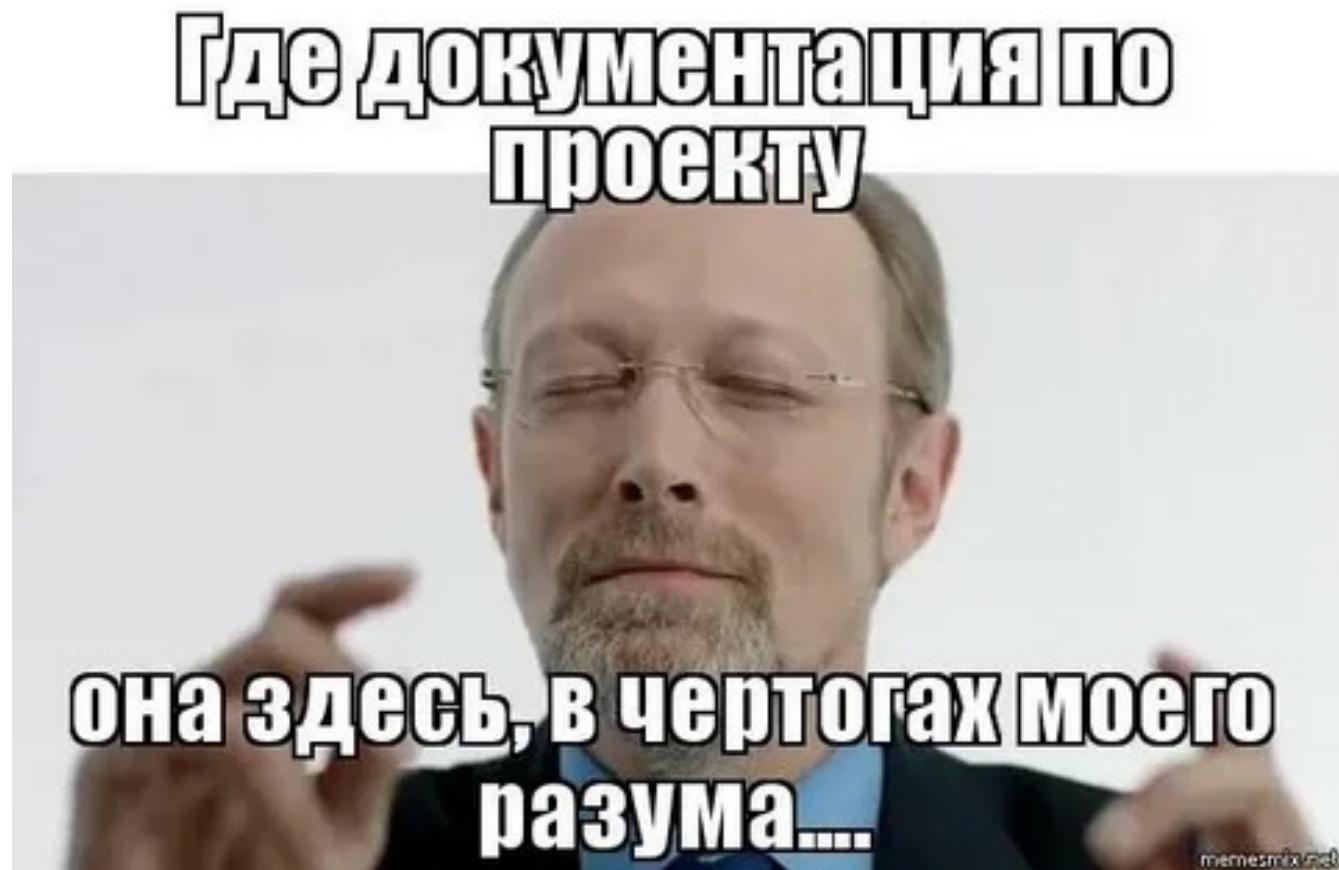
Минутка прекрасного



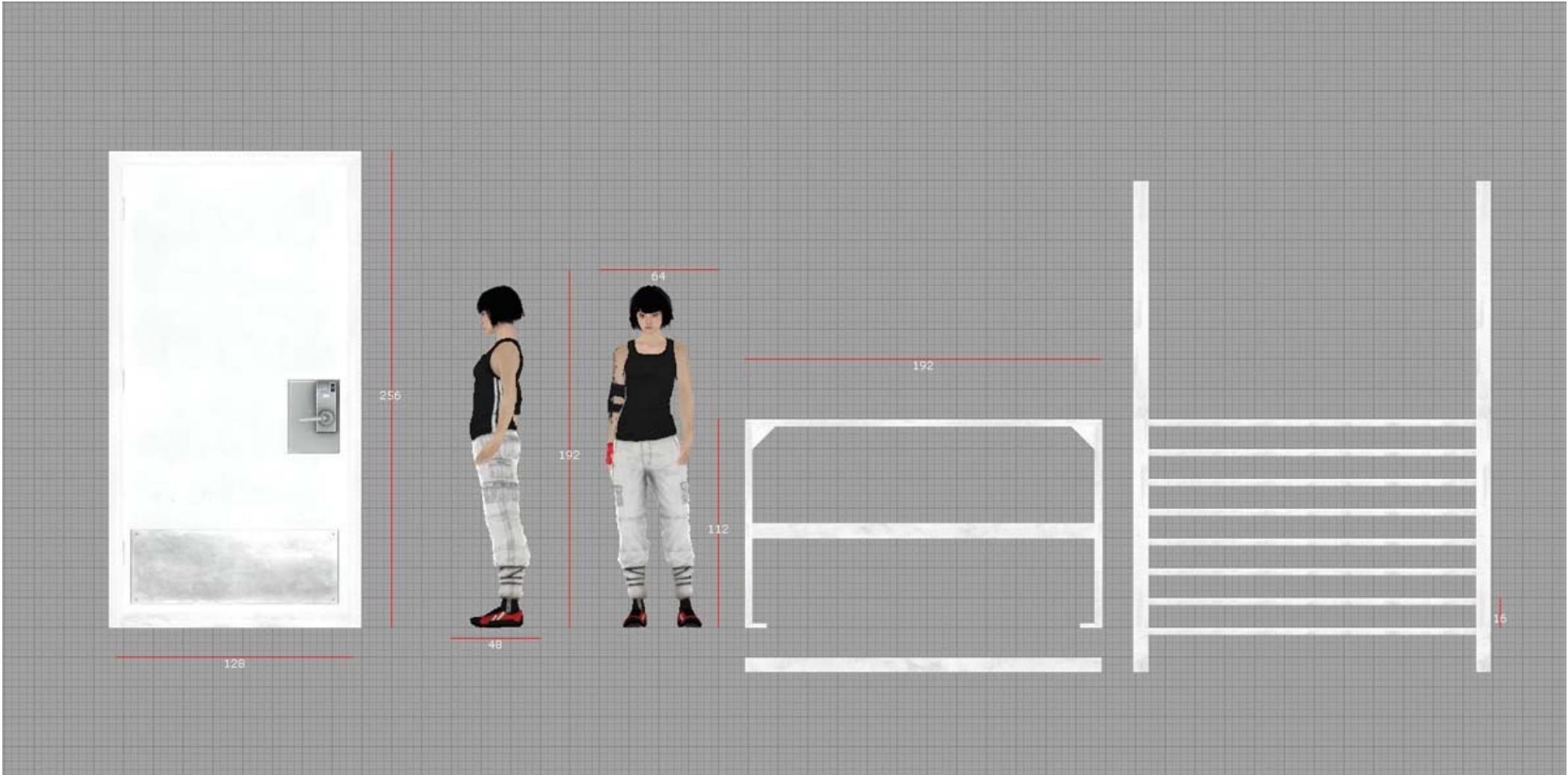
Планирование



Планирование геймплея и техническая документация



Стандарты игрового мира. Габариты персонажей



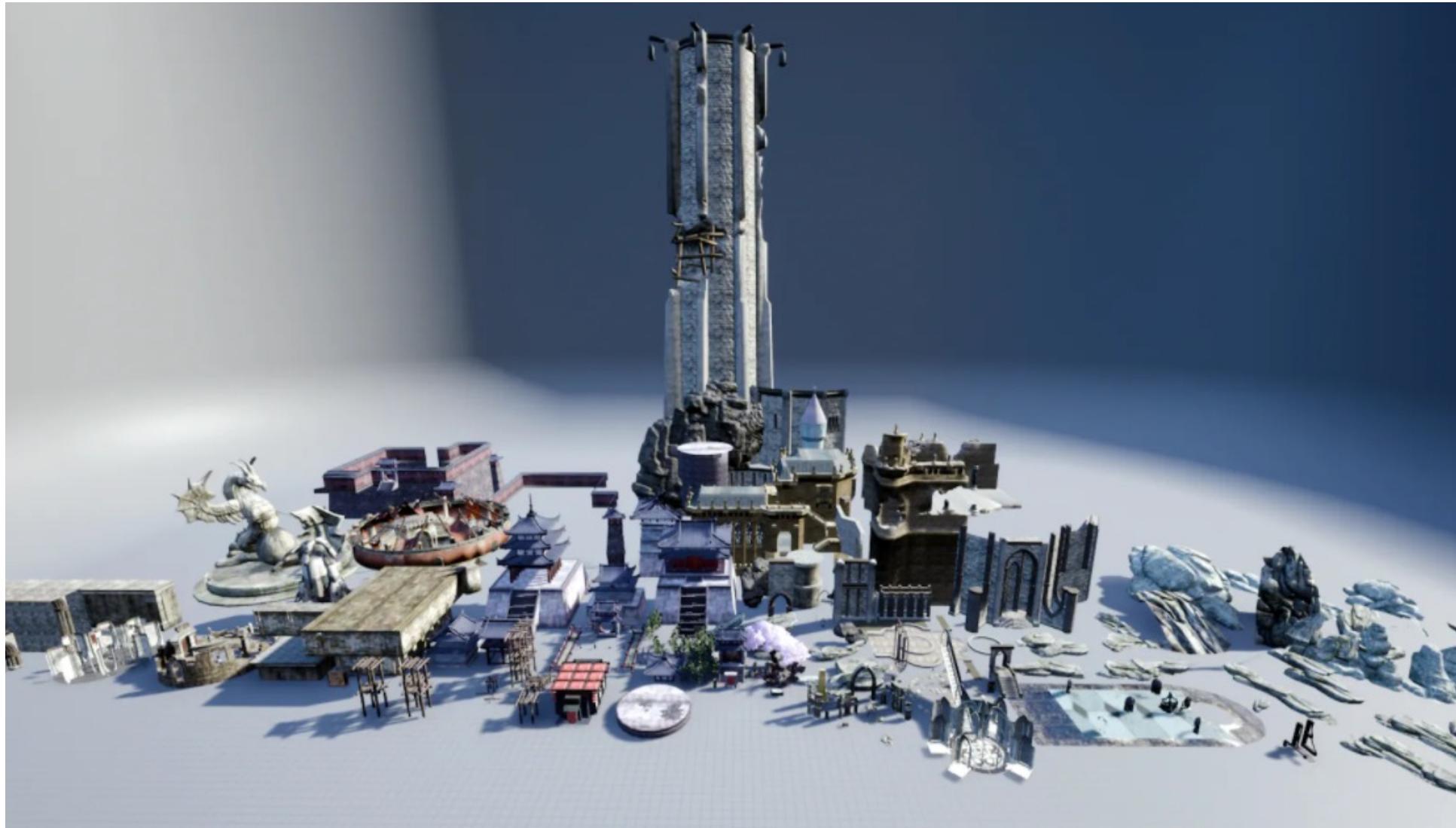
Прототипирование размеров игровой среды



Модульность в дизайне уровней.
Модульность в архитектуре



Принцип модульности



Подходы в разработке и оптимизации

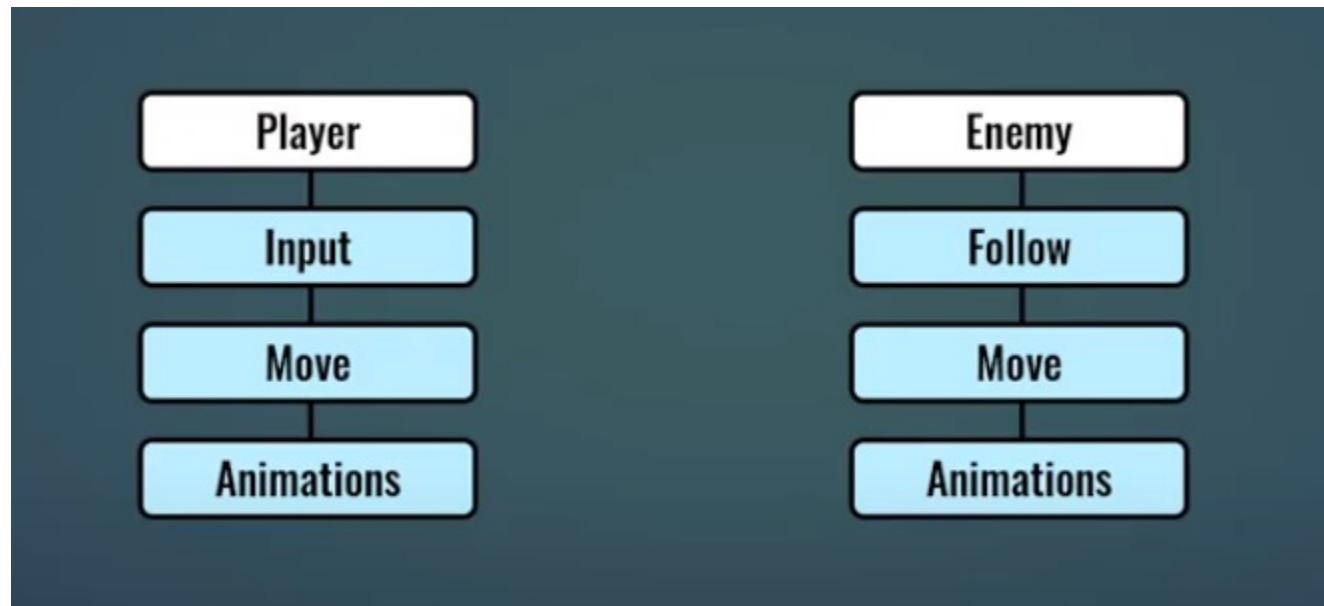
- Базовый или начальный подход
- Стандартный подход или Объектно-ориентированный

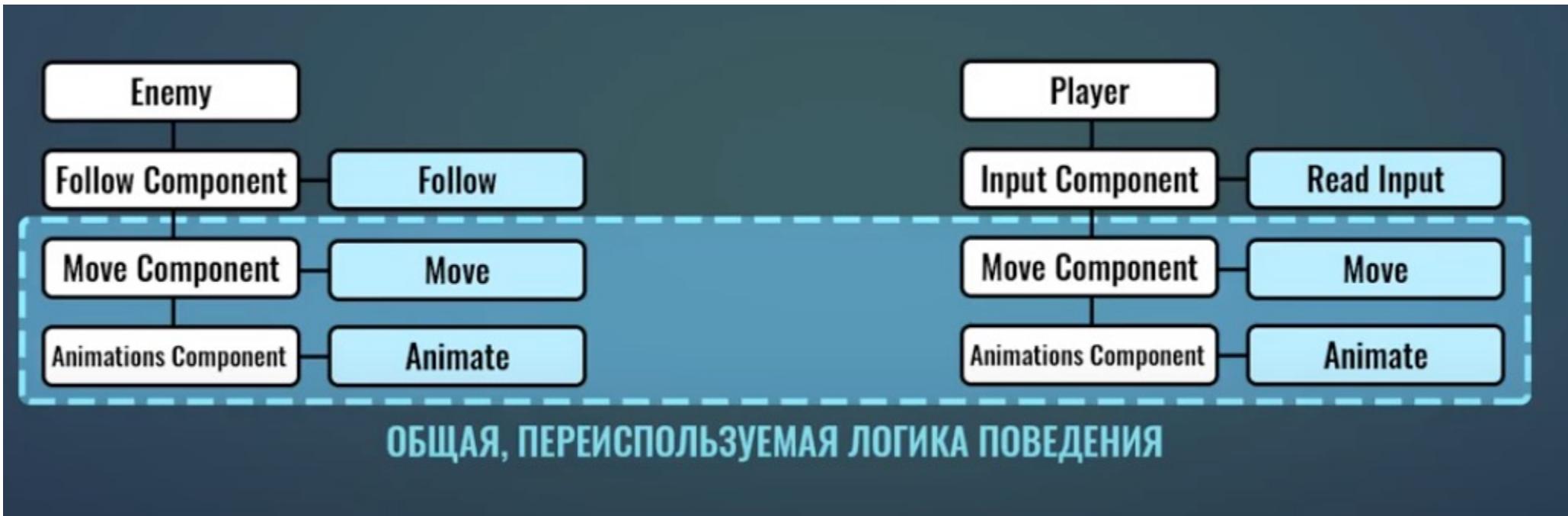
- Инкапсуляция — это способность класса скрывать свои внутренним компонентам от внешнего кода. Мы не можем обратиться к переменным (методам, процедурам и др.), объявленным с ключевым словом `Private`, `Protected` из внешнего, по отношению к классу кода, и это хорошо. Это позволяет добиться защищенности отдельных данных от внешнего проникновения, случайного или целенаправленного.
- Наследование — способность класса наследовать (копировать) методы, переменные, состояния и др. у своего родительского класса.

- Полиморфизм— способность функции обрабатывать данные разных типов. Данный подход позволяет, использовать параллельное программирование не , взаимодействовать сразу с несколькими типами объектов без указания их напрямую, обращаясь к тегу объектов, а так же переопределять метод другого скрипта. При таком подходе мы можем переменной присвоить экземпляр класса, так как он является наследником. Таким образом, скрипту будет все равно какой наследник класса лежит в переменной. Главное что у переменность есть метод.

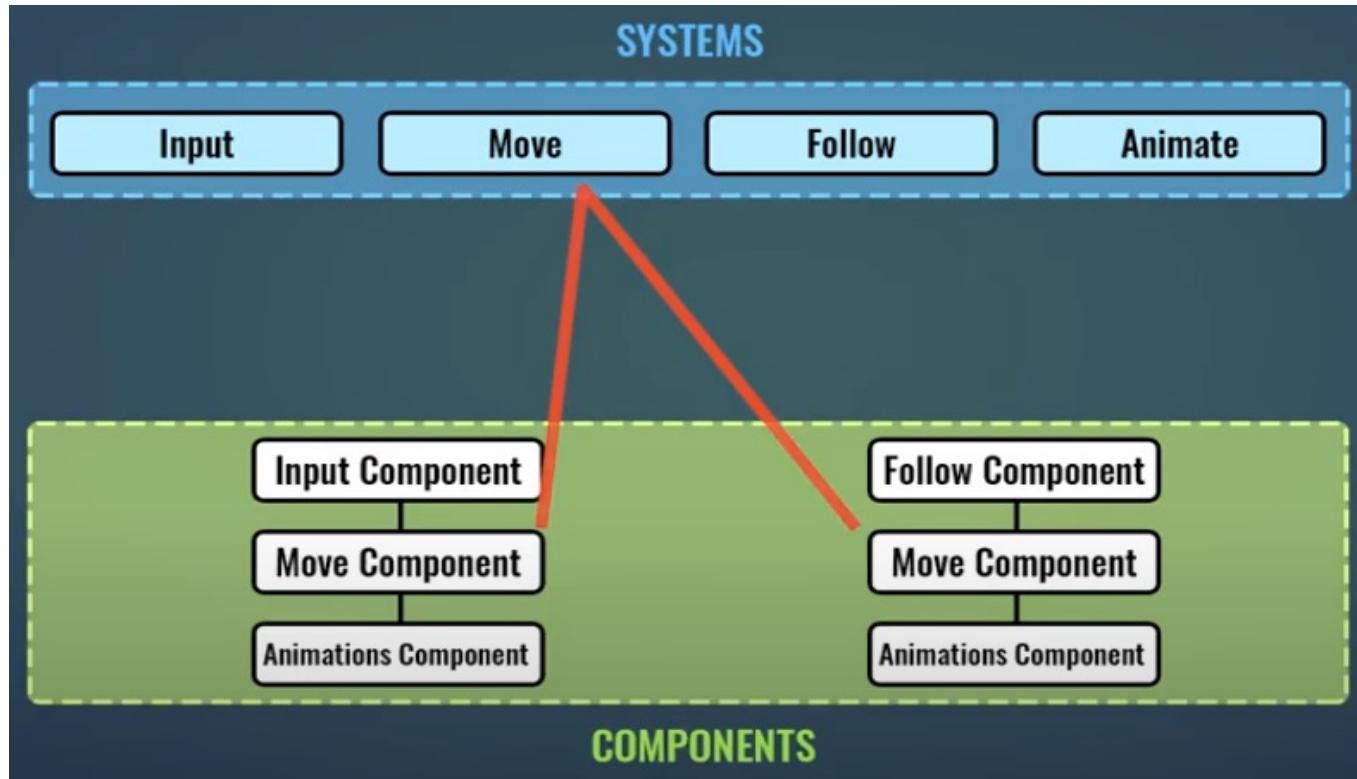
Примеры реализации структуры объектного подхода к реализации:

Данная архитектура удобна для простоты реализации, но имеет ключевой недостаток масштабируемость, в случае необходимости увеличения аналогичных объектов. Мы можем сделать наследование от одного из них, но это повлечет за собой ряд проблем и трудностей, в части кастомизации нового объекта.





ECS - Entity Component System



Ранее мы уже говорили про методы, скрипты и объекты, но основным делением при проектировании в условиях реализации ECS являются 3 понятия:

- сущности (Entities),
- компоненты (Components)
- системы (Systems)

Здесь мы подходим к ключевой цели – оптимизация используемых ресурсов и трудозатратность при реализации проекта.

ECS подход имеет бесспорное преимущество в случае использования в проектах большого числа однотипных компонентов сущностей. Но не является идеальным ввиду ряда факторов.

- Процесс разработки и реализации архитектуры системы, занимает больше времени в сравнении с классическим объектным подходом. Это компенсируется в случае расширения системы, позволяя более быстро ее дорабатывать.
- Проблематика читаемости кода – большее количество кода порождает большую вероятность совершения ошибок и возникновения багов. Дополнительные разбиения усложняют восприятие кода, в сравнении с минимальным количеством классов в MonoBehaviour

- UI и анимация – слабая сторона ECS. ECS как было сказано ранее основывается на обращении к параметрам, а не объектам. Из этого вытекает проблема нам придется и UI строить в том же data-oriented-стеке, отсутствие каких-либо фреймворков и наработок в этом направлении приводит к попыткам максимального упрощения UI и анимации.

Из всего вышеперечисленно выливается последняя проблема. Для решения ряда задач нам не обойтись без помощи MonoBehaviour, и взаимодействие двух координальных подходов накладывает ряд сложностей с их взаимодействием. ECS является лучшим решением в тех задачах где нужны только данные и логика. MonoBehaviour лучше в задачах визуального отображения и рендера.

Когда мы рассмотрели ECS подход к оптимизации и проектировании, то следует и рассмотреть принципы, которые следует придерживаться изначально, независимо от подхода к архитектуре. Оптимизация проекта это всегда многоуровневая задача. Разработчики Unity дают такие советы в направлениях оптимизации:

Рекомендации по CPU и GPU оптимизации:

- Не используйте треугольников больше, чем необходимо
- Объединяйте близко расположенные объекты: вручную или используя инструмент в Unity.
- Используйте меньше материалов, объединяйте текстуры в большие текстурные атласы.
- Используйте меньше объектов, которые должны визуализироваться несколько раз (отражения, тени, попиксельные источники света и т. п.).
- Страйтесь держать количество швов на UV-карте и количество жёстких рёбер (удваивающих вершины) как можно более низким
- Самое быстрое освещение — это то, которое не рассчитывается

Сри оптимизация

- Объединяйте объекты так, чтобы каждый меш содержал хотя бы несколько сотен треугольников и использовал только один Материал. Важно понимать, что объединение двух объектов, использующих разные материалы, не даст увеличения производительности. Основная причина, по которой два меша используют разные материалы, состоит в том, что они используют разные текстуры. Для оптимизации производительности CPU нужно убедиться, что объекты, которые вы объединяете, используют одну текстуру.

Графика и оптимизация

- Количество вершин, которое обрабатывает видеокарта, обычно не совпадает с количеством, показываемым 3D-приложением. Приложения для моделирования обычно показывают геометрическое количество вершин, то есть, количество угловых точек, составляющих модель. Для видеокарты некоторые геометрические вершины необходимо разбить на несколько логических вершин для корректной визуализации. Вершина может быть разбита на несколько, если она имеет несколько нормалей, UV-координат или вертексных цветов. Следовательно, количество вершин в Unity неизменно выше, чем количество вершин в 3D-приложении.
- В то время как количество геометрии в моделях оказывает влияние в первую очередь на GPU, некоторые функции Unity предполагают обработку моделей и на CPU

Самое быстрое освещение — это то, которое не рассчитывается. Используйте карты освещения для запекания статичного освещения вместо расчёта освещения в каждом кадре. Процесс создания карт освещения требует много времени, чем простое размещения источников света в сцене, но:

- Это намного быстрее работает (в 2–3 раза по сравнению с 2 пиксельными источниками света)
- Это выглядит лучше, так как вы можете запечь глобальное освещение и с более высоким качеством

- Пиксельное динамическое освещение добавит затраты на визуализацию каждого пикселя и может привести к появлению объектов, визуализируемых в несколько проходов. На маломощных устройствах, таких как мобильные устройства или дешёвые PC, следует избегать использования более чем одного пиксельного источника света, освещавшего каждый отдельный объект, и стараться использовать карты освещения. Вершинное динамическое освещение может добавить затраты для случаев вершинных трансформаций. Страйтесь избегать ситуаций, когда несколько источников света освещают один объект.
- Если вы используете пиксельное освещение, то каждый меш будет визуализирован столько раз, сколько пиксельных источников света его освещает. Если вы объедините два меша, находящихся далеко друг от друга, то это увеличит габариты меша. Все пиксельные источники света, освещавшие каждую часть объединённого меша, будут освещать теперь этот меш, поэтому количество проходов визуализации, необходимое для этого меша, увеличится. Как правило, число проходов для объединённого меша равно сумме проходов для составивших его частей, в результате чего нет выигрыша от объединения. По этой причине не стоит объединять меши, которые достаточно далеко друг от друга, чтобы быть освещёнными разными пиксельными источниками света.
- Использование сжатых текстур уменьшает размер ваших текстур (в результате они быстрее загружаются и занимают меньше памяти), а также может значительно повысить производительность

Использование мипмап для текстур

- Мирмар — это последовательность текстур, каждая из которых представляет собой поочередно уменьшенное представление того же изображения. Высота и ширина каждого изображения или уровня МИР-карты меньше вышестоящего уровня на число, соответствующее степени двух. Или другими словами метод текстурирования, использующий несколько копий одной текстуры с разной детализацией.
- В этом случае сжатие текстур поможет ограничить количество текстурных данных, транспортируемых в GPU при визуализации. Мипмапы позволяют GPU использовать для маленьких треугольников текстуры пониженного разрешения.
- Есть исключение из этого правила: когда один тексель (пиксель текстуры) соответствует одному пикселю экрана, что встречается в элементах пользовательского интерфейса и в 2D-играх.

Level Of Detail и Тени в реальном времени

- **LOD system** или **Уровни детализации** – снижение количества отображаемой геометрии или детальности объекта по мере удаления наблюдателя. Данный подход позволяет снять часть нагрузки в части рендера объектов, а для совсем малых объектов исключить их из дополнительной нагрузки в случае небольшого отдаления, без потерь в итоговом качестве картинки.
- Тени в реальном времени хорошо выглядят, но они могут сильно снижать производительность, одновременно добавляя дополнительные draw calls для CPU и дополнительную обработку для GPU. Данное использование не всегда целесообразно, а чаще всего необоснованно. Следует тщательно подходить к использованию данной технологии.

Операции математические и с плавающей точкой

- Трансцендентные математические функции (такие как pow, exp, log, cos, sin, tan) довольно ресурсоемки, поэтому по возможности избегайте их использования. Рассмотрите возможность использования текстур или аналогов функций в качестве альтернативы сложным математическим вычислениям, если это возможно.
- Избегайте написания собственных операций (таких как normalize, dot, inversesqrt). Встроенные опции Unity гарантируют, что драйвер может генерировать оптимизированный код.
- В то время как точность (float vs half vs fixed) переменных с плавающей запятой в основном игнорируется на настольных графических процессорах, очень важно добиться хорошей производительности на портативных графических процессорах. Не пренебрегайте отслеживанием и по возможности избегайте значений с плавающими точками.

- Порой сетевое решение (например, Photon) сильно снижает производительность. Программист видит, что из-за него происходят пики загрузки ЦП, но забывает заглянуть в стек вызовов. Сетевой инструмент запускает методы, вызываемые из сети (так называемые RPC), а они — часть вашего собственного кода, которая не имеет к сети никакого отношения. В таких ситуациях нужно оптимизировать RPC-методы и/или распределить их рабочую нагрузку.

Краткие рекомендации:

- Сохраняйте количество вершин между 200 000 и 3 000 000 в каждом кадре, если целевая платформа — PC
- Если вы используете встроенные шейдеры, проверьте категории шейдеров Mobile и Unlit. Они прекрасно работают и на немобильных платформах, но являются упрощёнными версиями более сложных шейдеров.
- Уменьшите количество различных материалов в сцене — используйте один материал для нескольких объектов, где это возможно.
- Установите свойство Static для неподвижных объектов, чтобы использовать внутреннюю оптимизацию static batching.
- Используйте сжатие текстур, когда это возможно, а также отдавайте предпочтение 16-битным текстурами перед 32-битными.
- Используйте только один (предпочтительно направленный) источник света, влияющий на вашу геометрию.
- Запечённое освещение всегда имеет преимущество перед динамическим освещением

- По возможности избегайте использования тумана.
- Узнайте преимущества технологии Occlusion Culling и используйте её для снижения количества видимой геометрии и количества draw calls в случаях со сложными статичными сценами с большим количеством перекрывающих друг друга объектов. Планируйте свои игровые уровни с учётом этой технологии.
- Используйте скайбоксы для имитации далеко расположенной геометрии.
- Используйте пиксельные шейдеры или инструменты для совмещения текстур, чтобы смешивать текстуры вместо многопроходной визуализации.
- Сводите к минимуму количество сложных математических операций в пиксельных шейдерах: pow, sin, cos и т. п.
- Используйте меньше текстур.
- Используйте переменные половинной точности, где это возможно.
- Внимательно изучайте инструменты и ассеты, которыми пользуетесь.

Современные технологии в компьютерной графике

Hidden surface determination

При рендеринге в компьютерной графике возникает вопрос, какие части поверхностей видны в 3D-сцене при проецировании на двумерную поверхность дисплея. Для ускорения вычислений и уменьшения нагрузки был предложен алгоритм позволяющий отключать скопление объектов, которые находятся вне поля зрения игрока. Стоит так же отметить что данная идея подразумевает исключение из рендера не только объектов находящихся в непосредственно за спиной наблюдателя, но и скрытия объектов, видимость которых скрывают другие объекты сцены. На данный момент реализованные 2 такие технологии Frustum Culling и Occlusion Culling поддерживаются и в Unreal engine, и в Unity.

Методы масштабирования разрешения

Изменение размера цифрового изображения с сохранением пропорций. Суть метода в приведении изображения низкого разрешения в более высокое. Выделяется несколько технологий DLSS, Nvidia Image Scaling, AMD FSR, билинейная или бикубическая фильтрация.



NATIVE 1440P

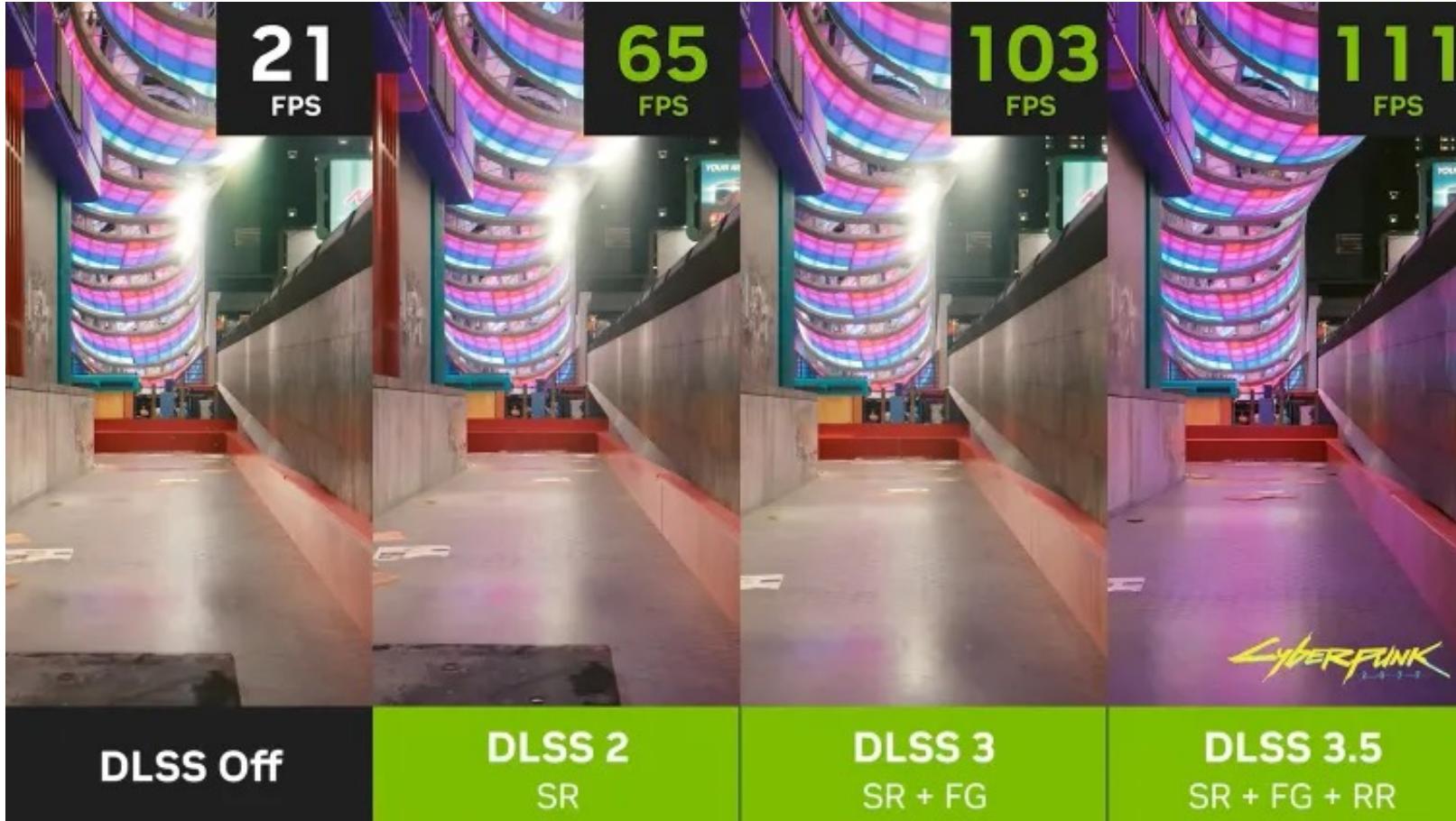


SPATIAL UPSCALING 1440P
ULTRA QUALITY MODE



DLSS 1440P
QUALITY MODE

DLSS



Прирост в производительности показывают разные версии апскейлинга.

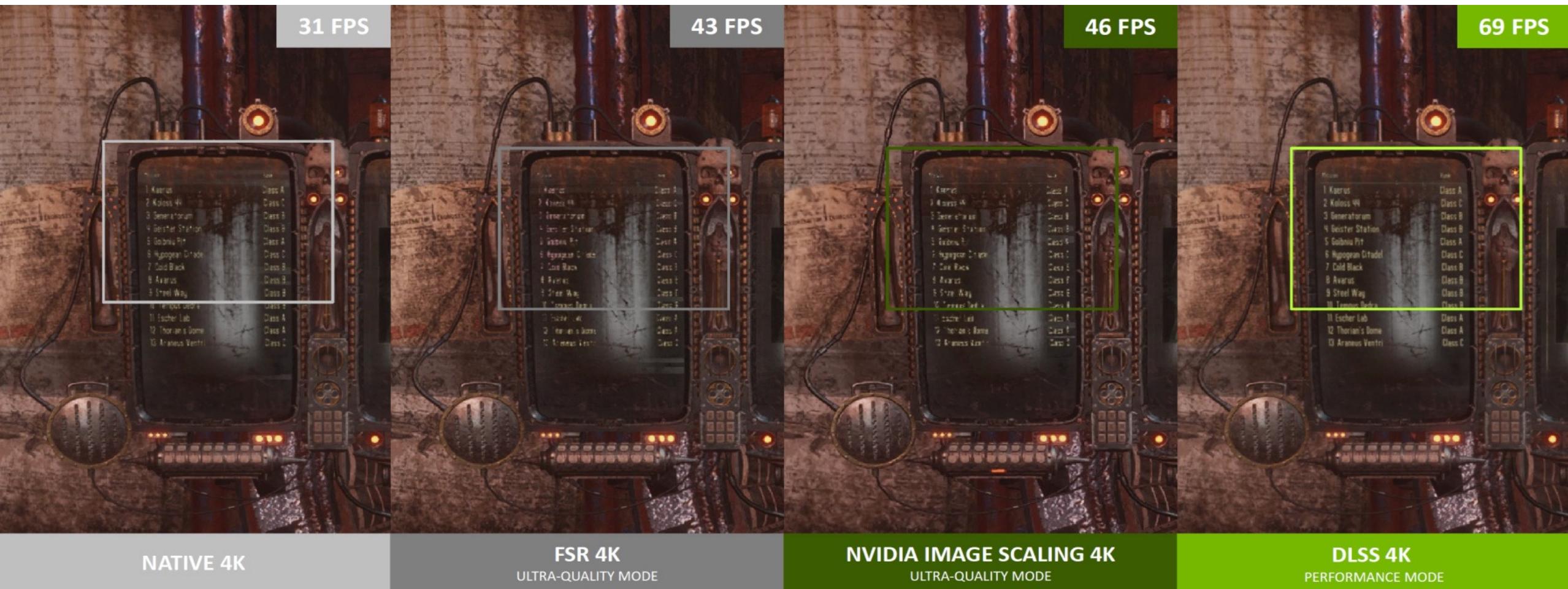
Проблема масштабирования - Гоустинг



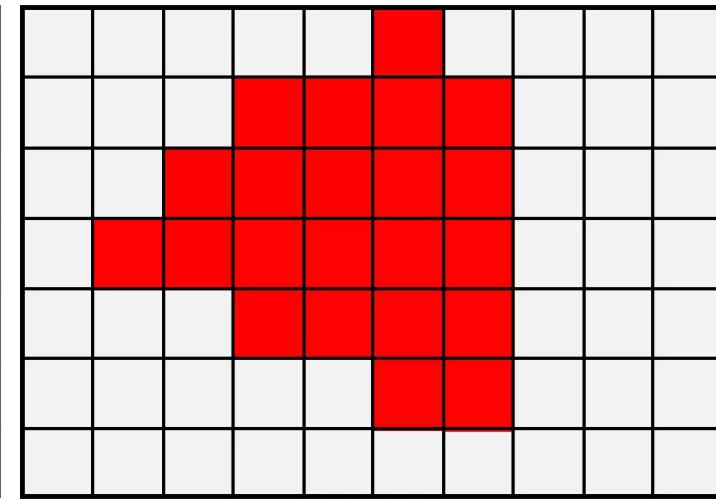
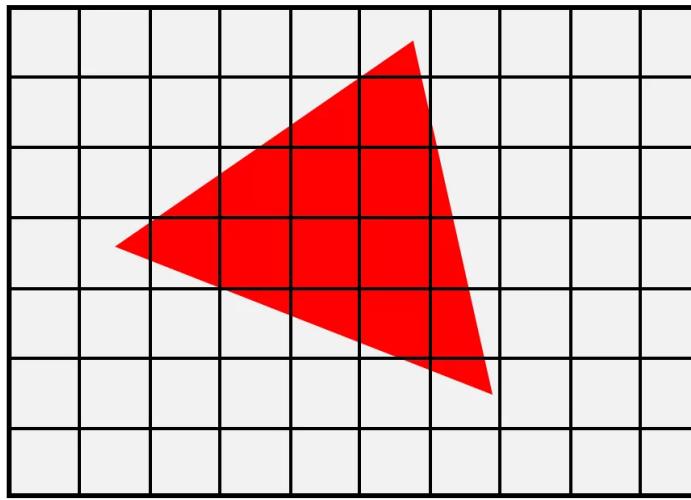
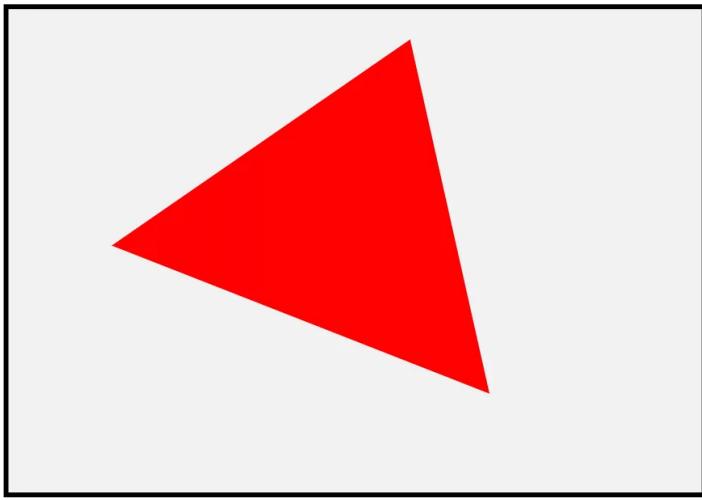
При использовании техник пространственного масштабирования (FSR и NIS) на этом примере хорошо заметна потеря детализации на тонких объектах, вроде проводов, антенн, сеточных заборов и другой подобной геометрии. Если временное масштабирование DLSS сохраняет такие детали за счет большего количества информации, взятой из предыдущих кадров, то техники пространственного масштабирования зачастую их портят или вовсе не отрисовывают — особенно на большом расстоянии.

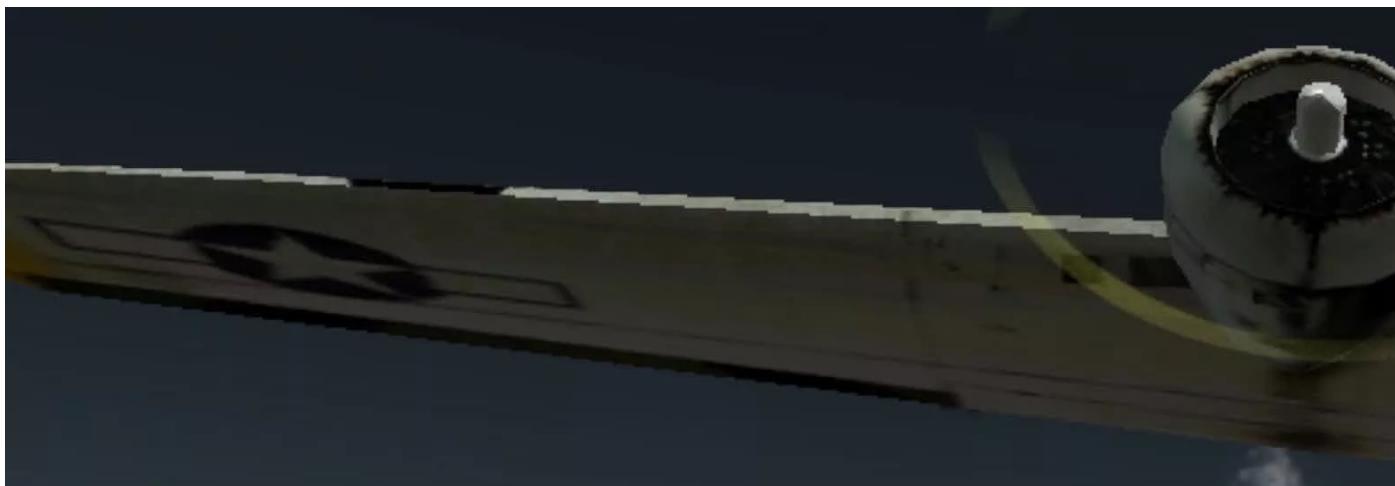
Еще один вид артефактов, выявляемый при использовании техники пространственного масштабирования и фильтра повышения резкости — свечение (ореолы) вокруг некоторых граней, возникающие из-за слишком большого повышения локальной контрастности, например когда темные объекты отображаются на светлом фоне. Это перекликается с еще одним побочным эффектом пространственного масштабирования

FSR / NIR / DLSS



Технологии сглаживания



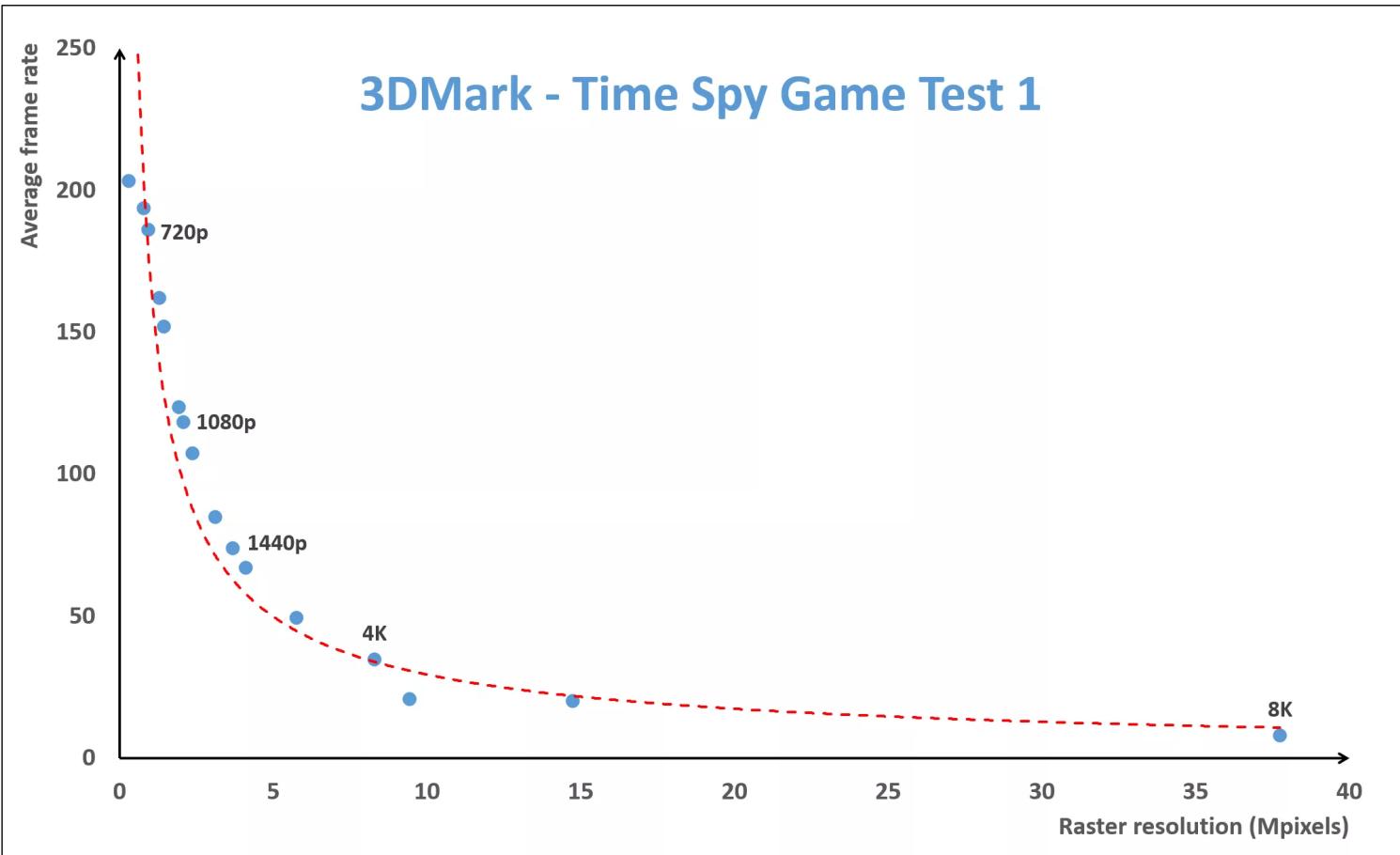




3DMARK 03

FPS:

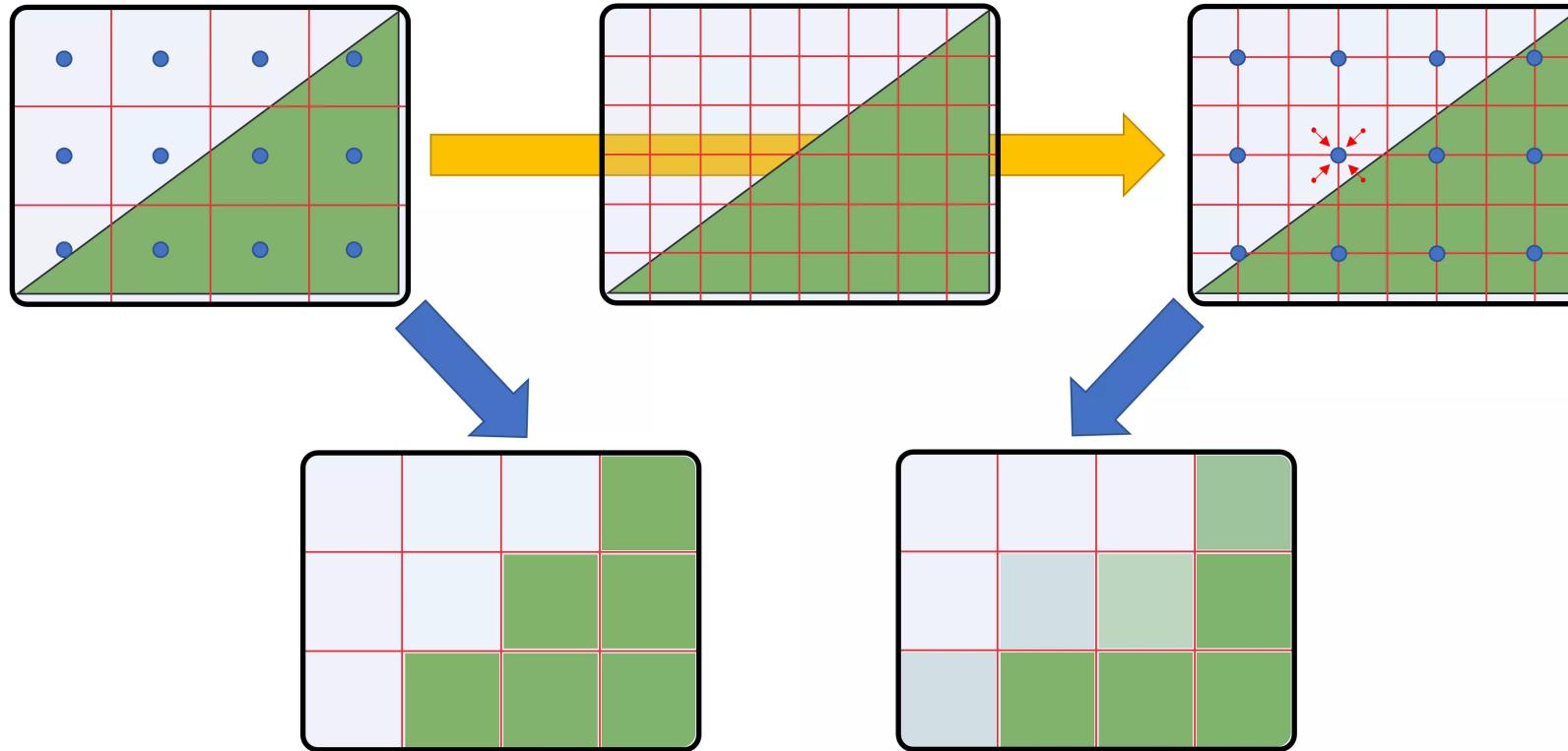
Time: 0:34.63 | Frame: 1038



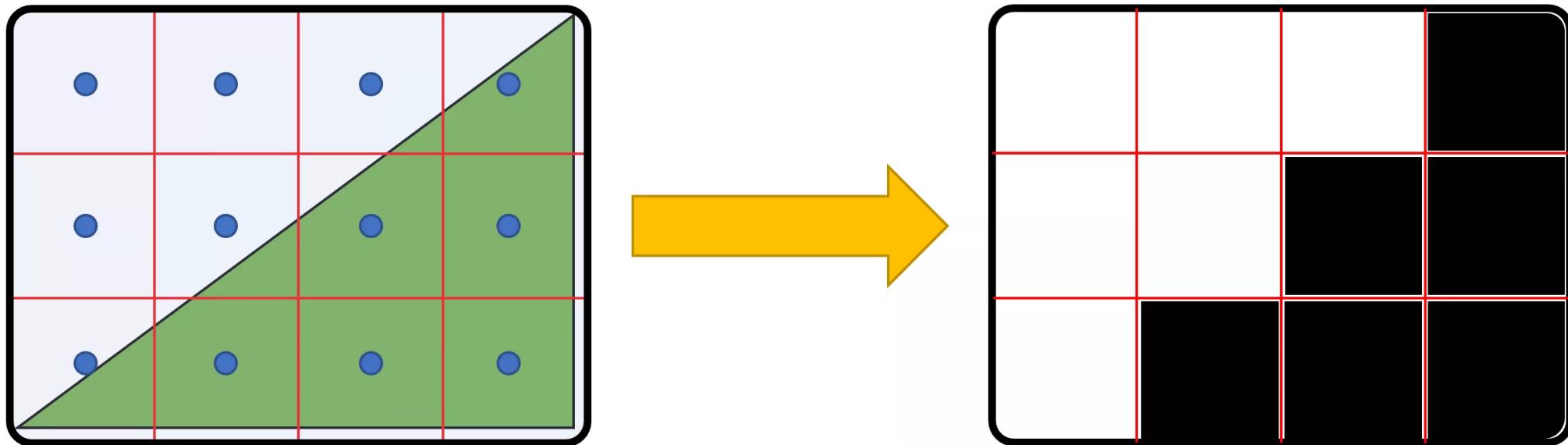
Технологии сглаживания

- Multisample Anti-Aliasing (MSAA)
- Supersample Anti-Aliasing (SSAA)
- Fast Approximate Anti-Aliasing (FXAA)
- Temporal Anti-Aliasing (TAA)

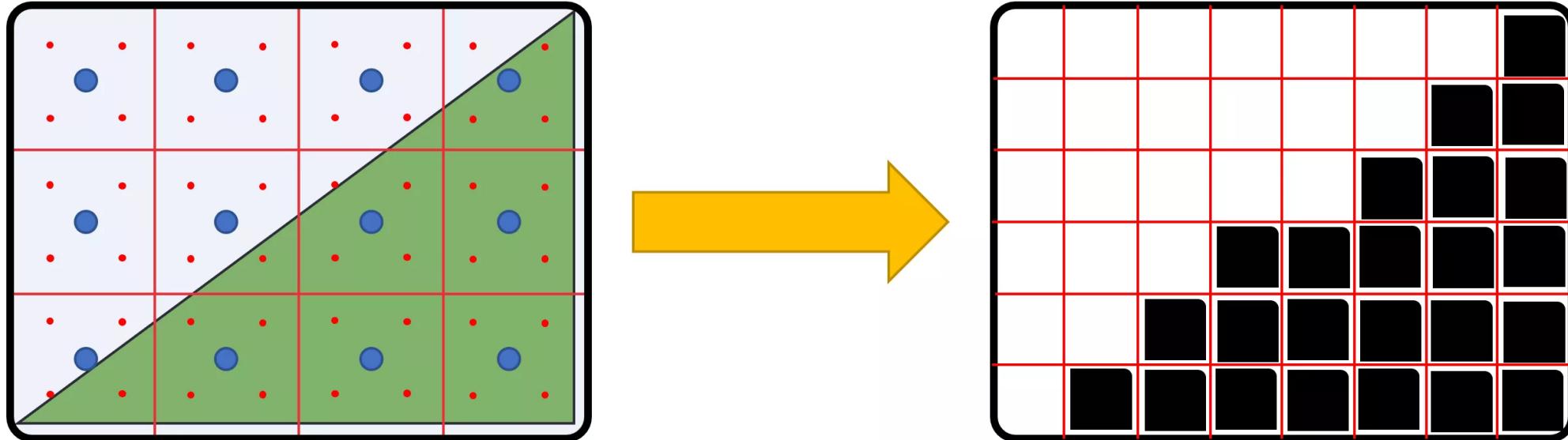
Supersampling anti-aliasing (SSAA)



Multisample anti-aliasing (MSAA)



Multisample anti-aliasing (MSAA)



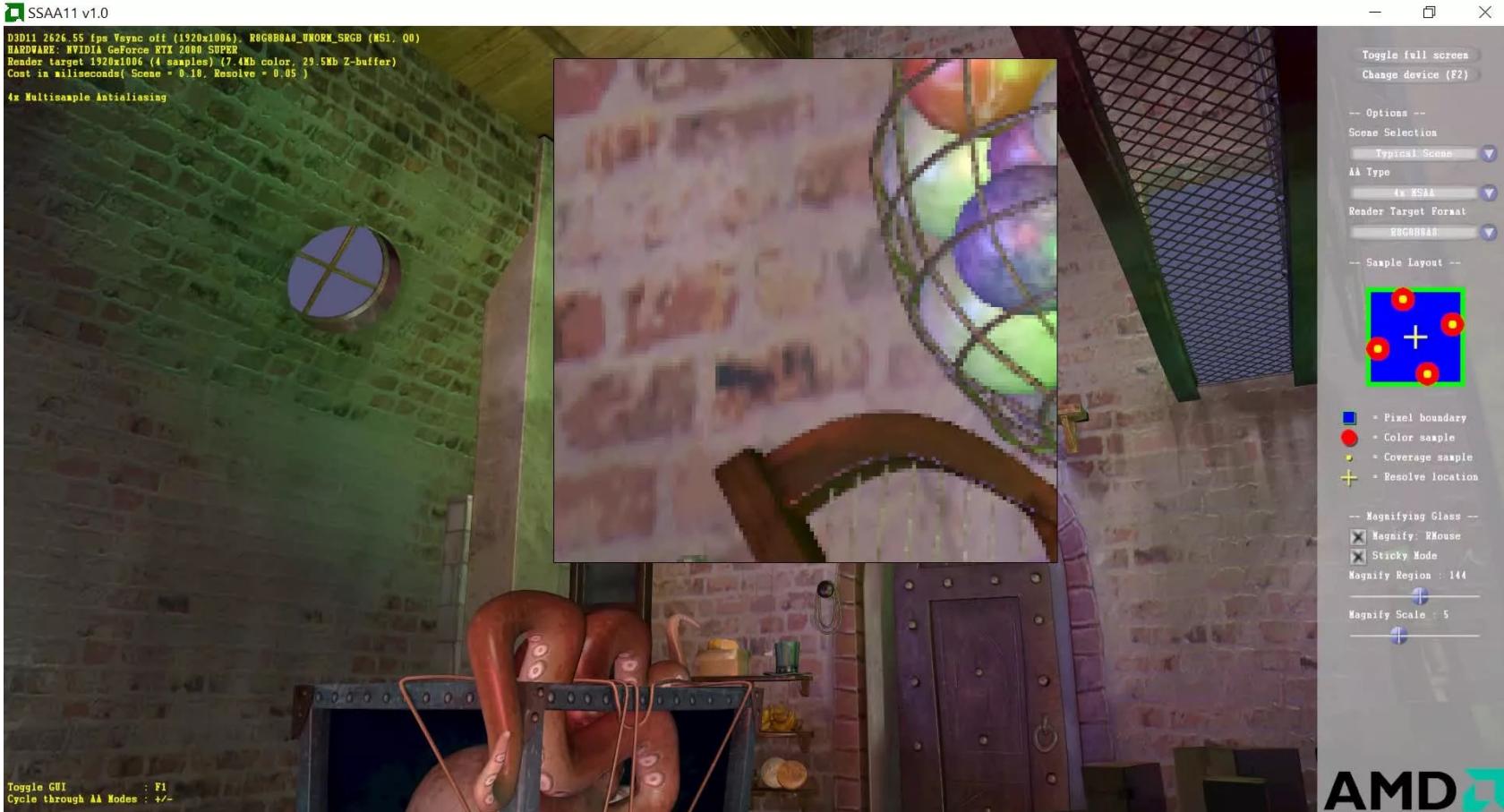
Multisample anti-aliasing (MSAA)



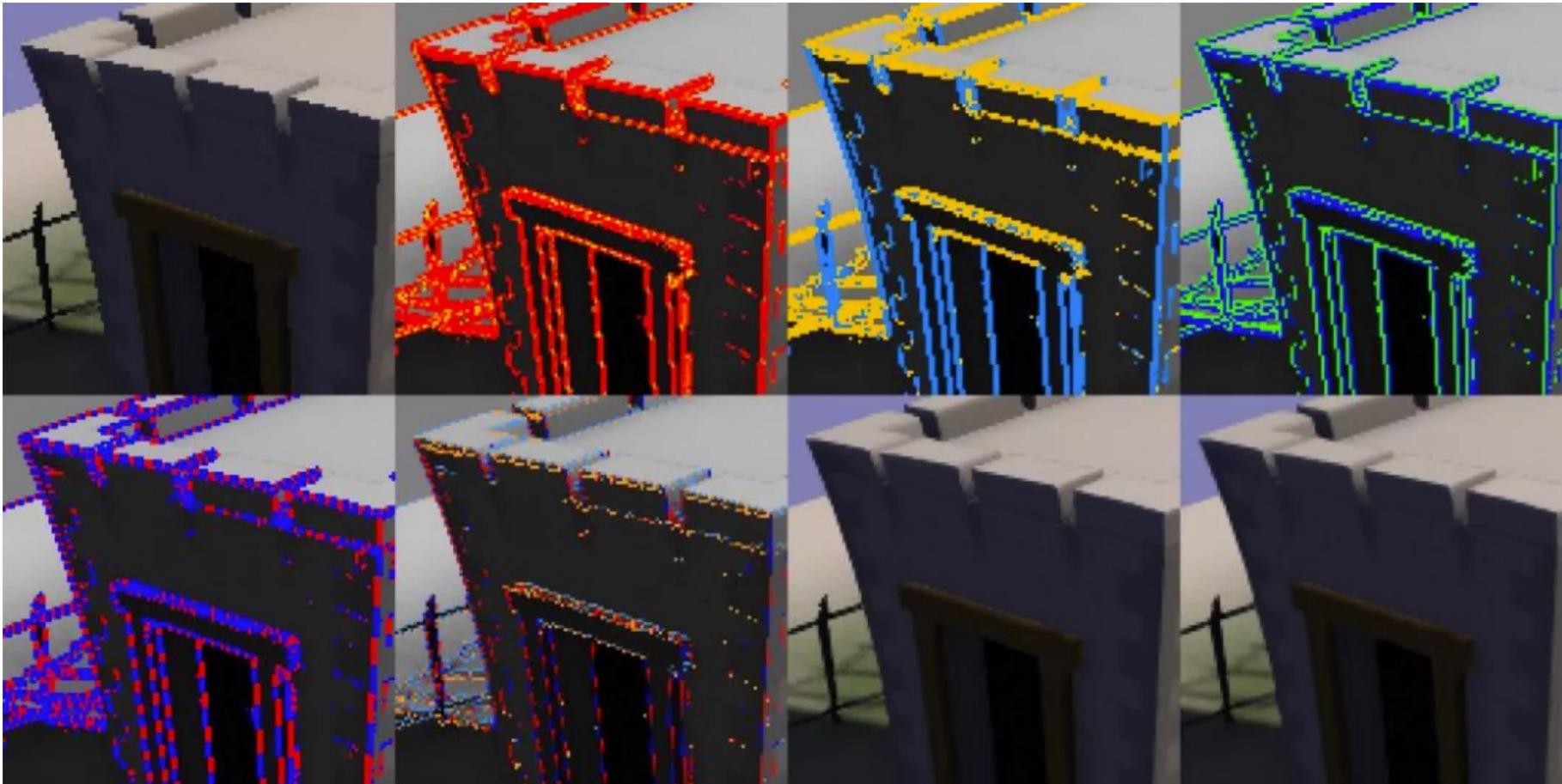
Multisample anti-aliasing (MSAA)



Multisample anti-aliasing (MSAA)



Fast approximate anti-aliasing (FXAA)

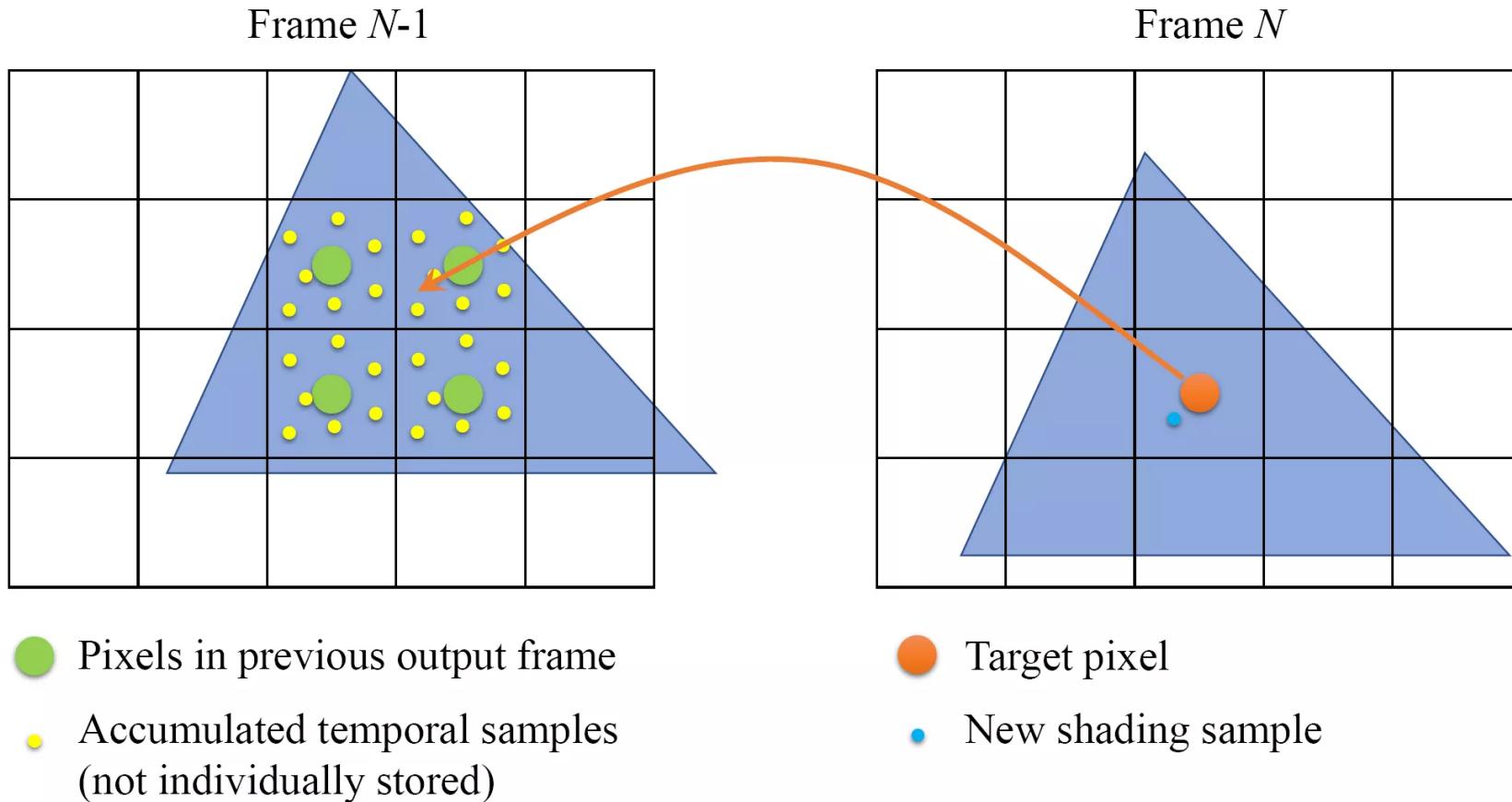


Fast approximate anti-aliasing (FXAA)

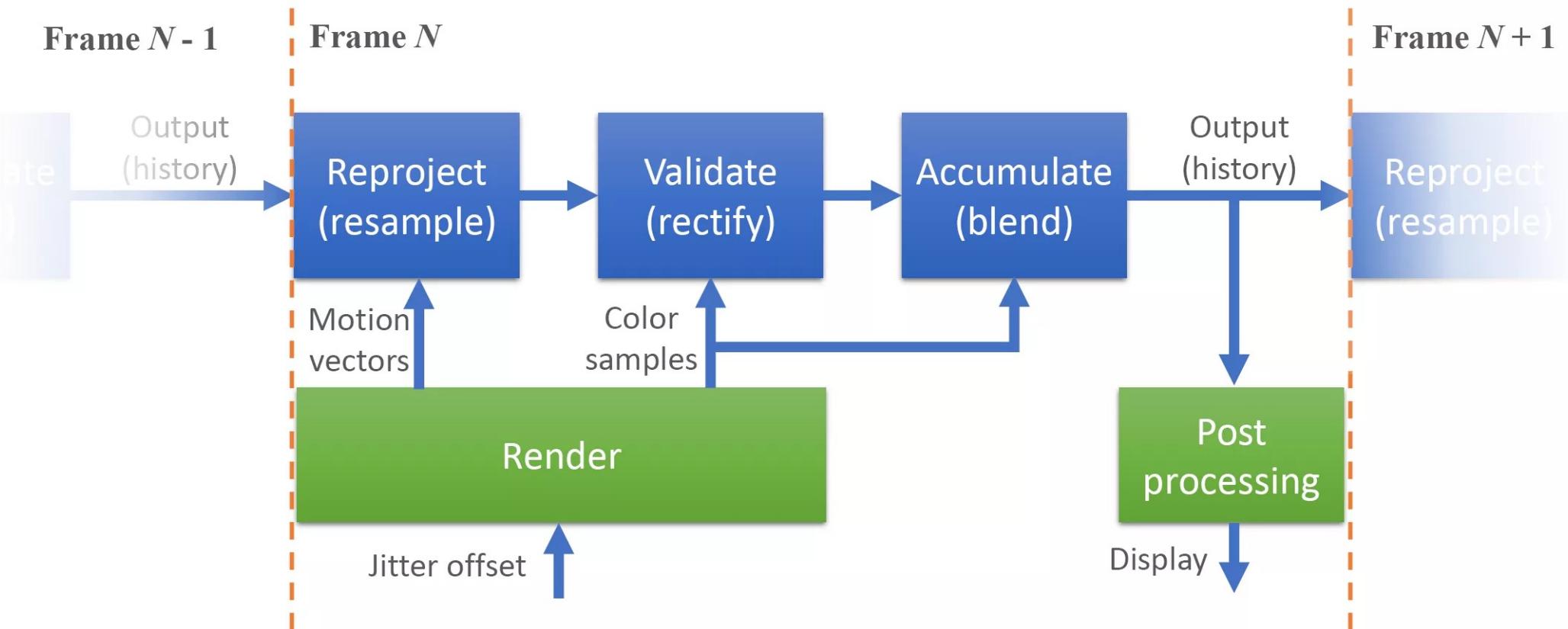


Без АА (слева) и FXAA (справа)

Temporal anti-aliasing (TAA)



Temporal anti-aliasing (TAA)



Temporal anti-aliasing (TAA)



Без АА (слева) и ТАА (справа)

DLSS



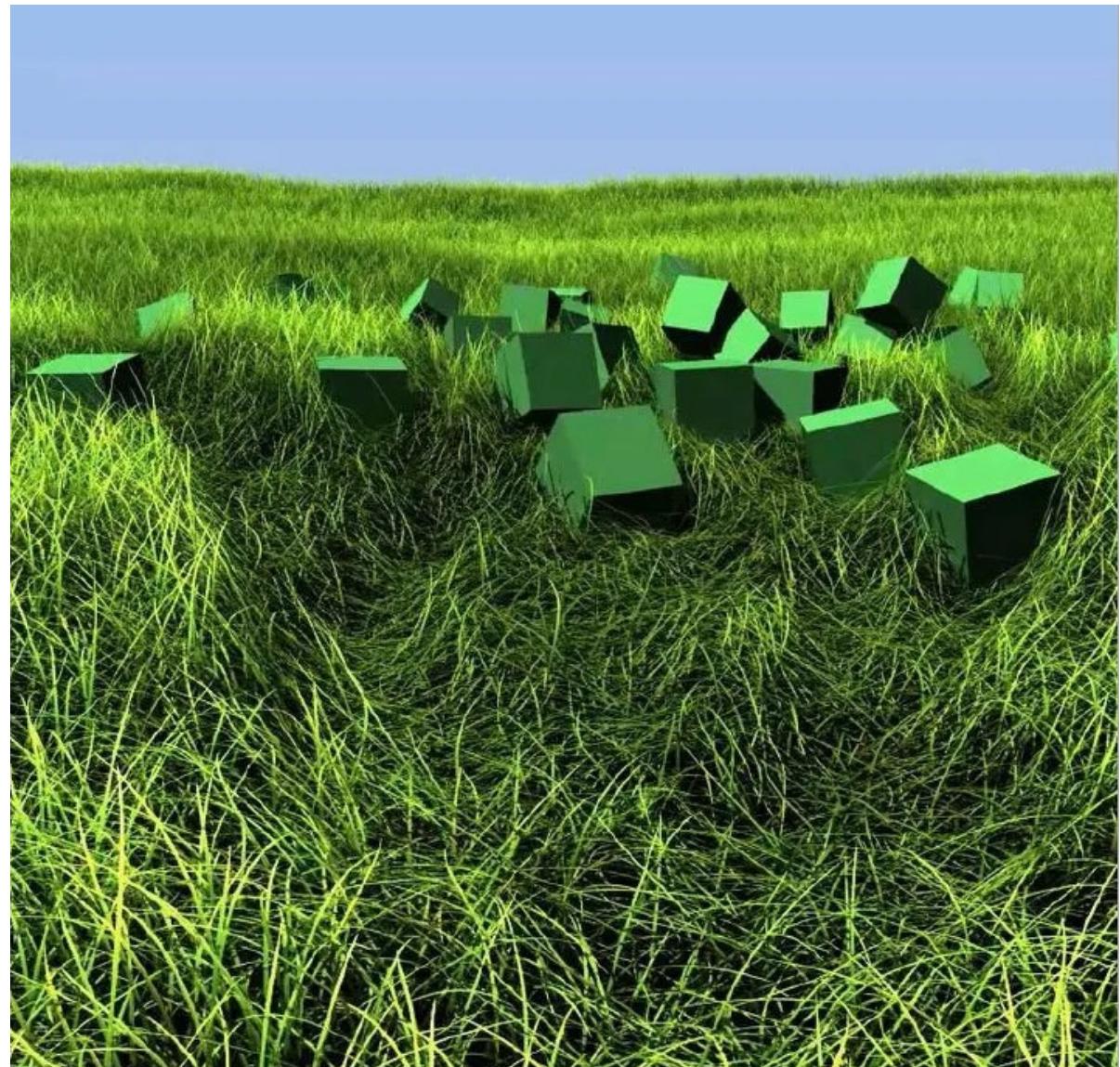
DLDSR



DLDSR справа, DSR слева.

Turf Effects (Nvidia)

Технология симуляции и отображения сложной физической растительности, разработка которой на данный момент находится в замороженном состоянии, позволяет осуществлять рендер многомилионной пиксельной растительности на сцене.



Ray tracing (Трассировка лучей)

Трассировка лучей — это общий метод геометрической оптики моделирования пути, по которому проходит свет, следя за лучами света при их взаимодействии с оптическими поверхностями. Но в последнее время данный метод просчета реалистичного освещения смогли адаптировать под использование в реальном времени. Преимущества для игровой индустрии заключаются в:

- Высокой реалистичности полученного изображения
- итоговая картинка имеет более высокого качества, по сравнению с растеризацией
- позволяет получить более-менее похожие на правду отражения при некоторых ограничениях
- рендеринг прозрачных и полупрозрачных объектов

Ключевым недостатком данной технологии является большие требования к системным требованиям для использования технологии

Гибридный рендеринг

Как было сказано ранее RT требует больших ресурсов, ввиду этого на данный момент развивается и совершенствуется гибридный рендеринг, смешения двух методик растеризации и трассировки. К примеру, основу геометрии можно растеризовать с высокой производительностью, а затем при помощи трассировки лучей просчитывать только мягкие тени и отражения. Хотя растеризация продолжит играть важнейшую роль и в ближайшие годы с появлением гибридного рендеринга, доля алгоритмов трассировки лучей в таких движках будет постепенно расти исходя из роста вычислительных возможностей будущих GPU.

Спасибо за внимание!



ИСТОЧНИКИ

- Сглаживание с помощью SSAA, MSAA, FXAA, TAA и других методик
-<https://habr.com/ru/articles/558552/>
- DLDSR <https://dtf.ru/games/2359047-poyasnyayu-za-dldsr-i-rasskazyvayu-kak-ego-pochinit#1>