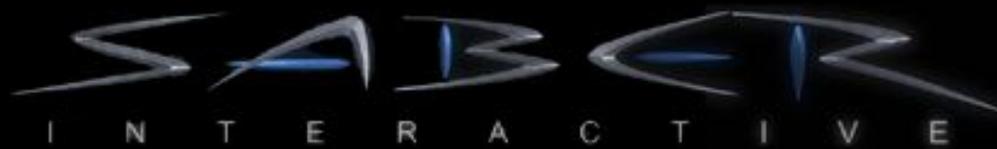




SABER
I N T E R A C T I V E

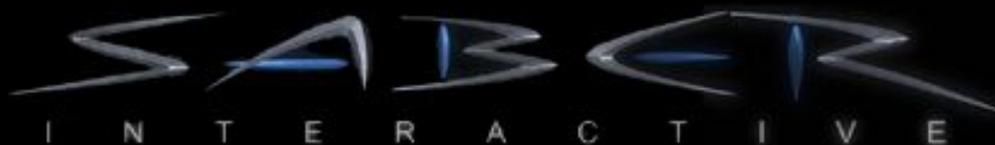
Промышленные Über-shaders





Шейдинг-система. Требования.

- Богатые возможности реализации материалов
- Поддержка нескольких механизмов освещения
- Максимальная производительность без скачкообразных провалов
- Мультиплатформенность с возможностью локальной специализации
- Минимальный объем памяти для ресурсов



uber shader. Один шейдер.

- По шейдеру на фичу материала / схему лайтинга
 - Сложно программировать и поддерживать
 - Неудобно выбирать шейдер в рантайме
- Мультипасс лайтинг
 - Vertex processing and memory bandwidth overhead
- Один код, управляемый константами
 - Computational overhead (many lights, SM, POM)
 - Все равно по отдельной реализации на платформу



Uber shader. Дефайны КОМПИЛЯЦИИ.

- Почему не статик бранчинг?
 - Рантайм-изменение константы ведет к перекомпиляции
- Почему не динамик бранчинг?
 - Вычислительный оверхед
 - Аппаратные ограничения



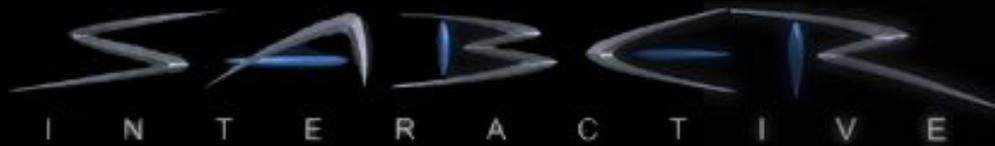
Uber shader. Комбинаторика.

Material (54)

Lighting (16)

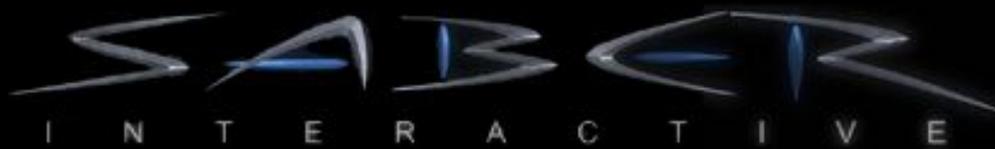
Platform (8)

Render pipeline (5)



Компиляция по требованию.

- Очевидная и простая реализация
- Редактирование шейдеров «на лету»
- Не используется ни одного лишнего шейдера
- Значительные ран-тайм провалы
- Неопределенные аллокации в компиляторе



Компиляция по требованию + КЭШ.

- Runtime stall только при первом обращении к шейдеру – может быть терпимо в некоторых случаях.
- При усложнении шейдеров даже одноразовые провалы критичны (компиляция до 20 сек, до 150 шейдеров в кадре)!
- Нужна система подготовки кэша для сборки.



Ручная сборка кэша.

- Тестовые «насыщающие» забеги
 - 9 x 16 платформ (16 = 4 бинарные опции)
 - 40 уровней
 - Инвалидация кэша при изменении уровня
 - Инвалидация кэша при изменении кода шейдера
- Собирать бинарный кэш для билда таким образом - неподъемная задача



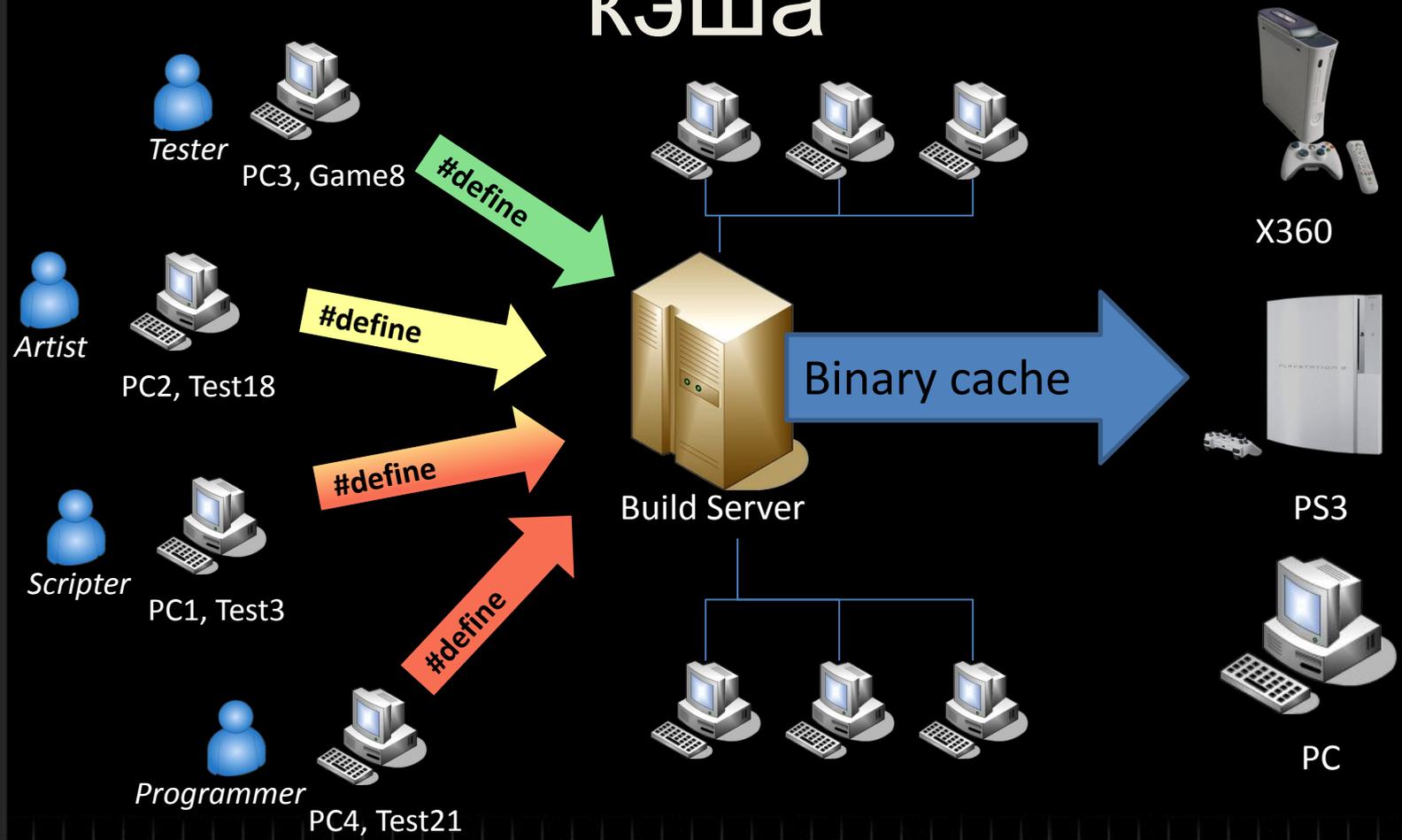
Полуавтоматическая сборка кэша

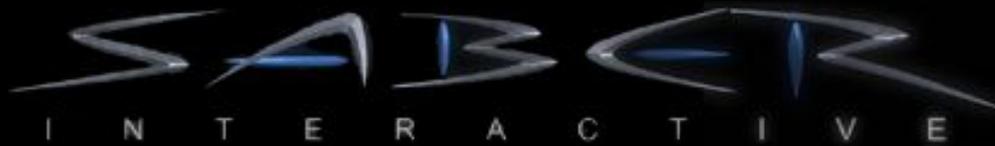
- Четко разделить параметры платформы / видео-опций от всех остальных и собирать независимые параметры компиляции
- Обеспечить механизм аккумуляции результатов игры со множества машин
- Обеспечить компиляцию бинарного кэша для разных платформ



Полуавтоматическая сборка

кэша





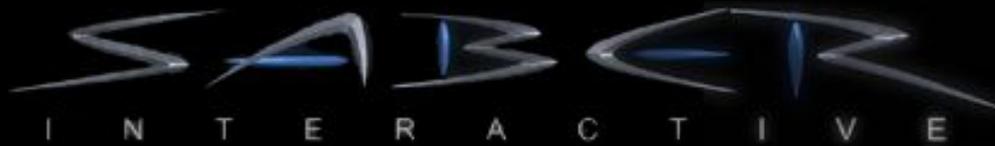
Полуавтоматическая сборка кэша

- Отсутствие промахов не обеспечивается
- Инвалидация при изменении уровня
- Инвалидация при изменении набора дефайнов
- Большая избыточность
- Большая сложность сборки per-level кэша для консолей



Автоматическая сборка кэша. Пререндер.

- Замена «тестера» в п/автоматической схеме
- Легкий ребилд при изменении арта
- Отсутствие избыточности, хорошее покрытие
- Неполное покрытие
- Сложность расстановки контрольных точек
- Необходимости поддержки прочих



Автоматическая сборка кэша. Интеллектуальный перебор.

- Полное покрытие
- Никакой дополнительной работы / данных
- Фиксирование параметров материалов, реструктуризация кода рендеринга
- Реализация и поддержка механизма перебора параметров
- Некоторая избыточность (КПД < 0.5)



SABER
I N T E R A C T I V E

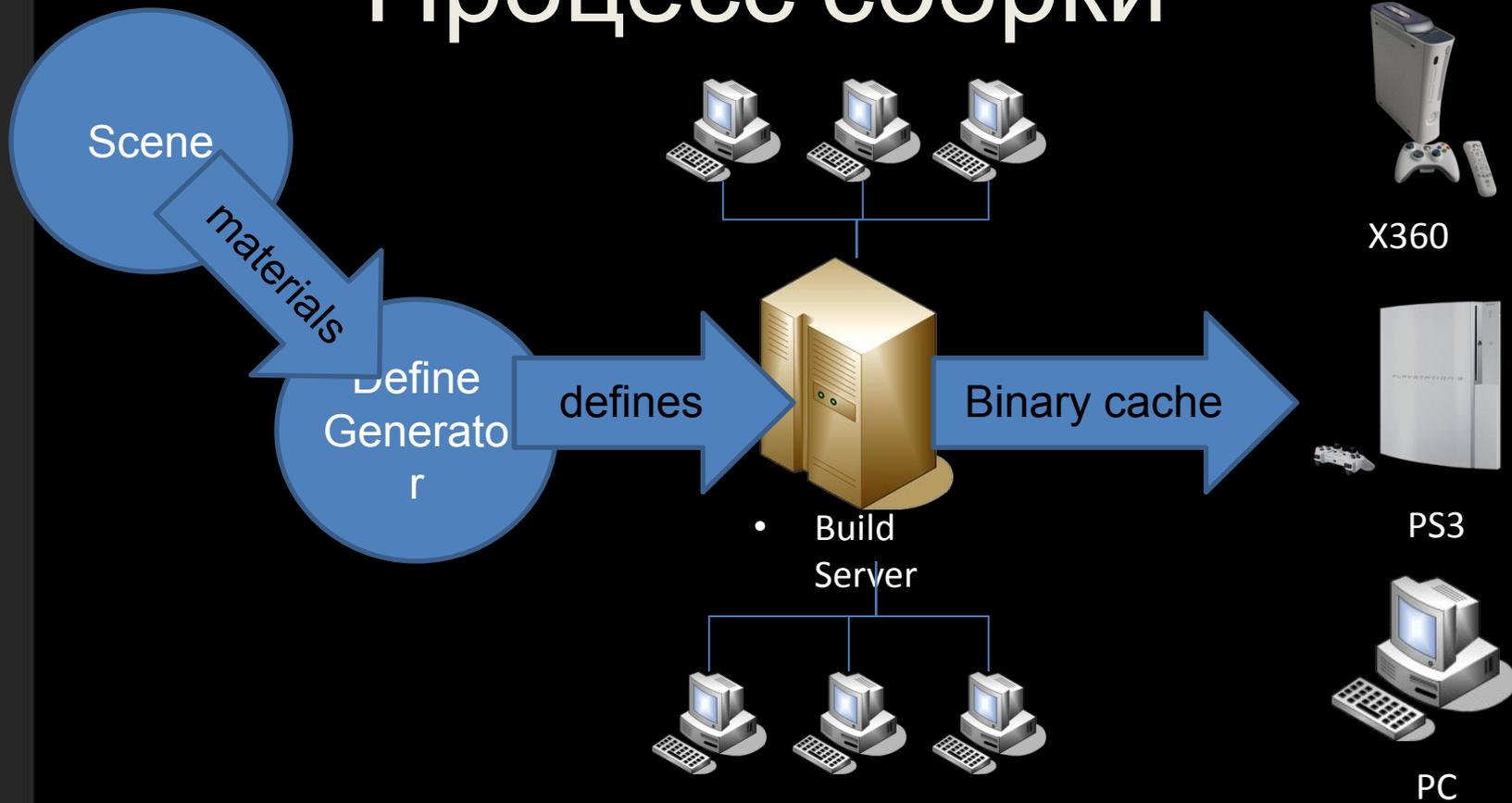
Автоматическая сборка кэша. Интеллектуальный перебор.



CACHE,
> 500k total



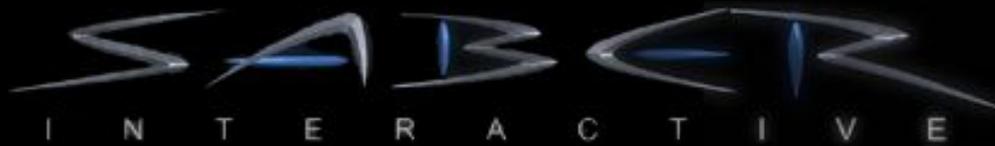
Процесс сборки





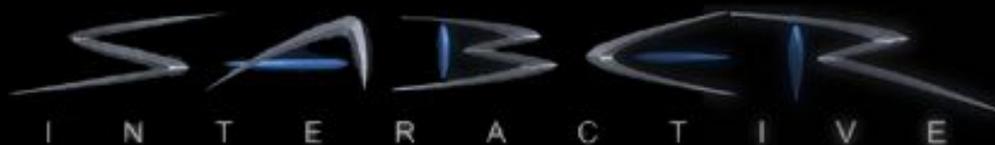
Сокращение кол-ва комбинаций.

- Дополнительные вычисления
 - Все динамические источники – прожекторы
 - Туман всегда включен
- LOD сделать статической опцией
- Dynamic branching для динамических источников (PC, Hi-LOD only)
- PC: 25k shaders total (13k/level total, 4k/level)
- X360: 9.5k shaders / level



Runtime, КОНСОЛИ

- Кэш хранится per-level, загружается полностью вместе с уровнем
- Весь кэш в памяти в компрессированном виде
- Шейдеры распаковываются и создаются по мере необходимости
- Макс 5000 шейдеров одновременно, LRU
- CreateShader() работает быстро, нет существенных провалов



Runtime, PC

- За D3D Runtime работает драйвер
- CreateShader() работает медленно
либо
- Первый DIP с только что созданным шейдером работает медленно
либо
- И то и другое + зависимость от стейтов 😊



Runtime, PC