



The Creation of Saints Row's Open World Cityscape: Stilwater



The Creation of Saints Row's Open World Cityscape: *Stilwater*

- 정리:김성익
- 2007/08/26



WWW.GDCONF.COM



컨셉

- **Stilwater** 개발 프로세스
 - 프로덕션 프로세스
 - 툴
 - 기술적인 관점
- 새로운 도전
 - 대형 오픈 월드형 게임
 - 개발 경험이 없는 장르의 게임
 - 새로운 엔진과 새로운 플랫폼

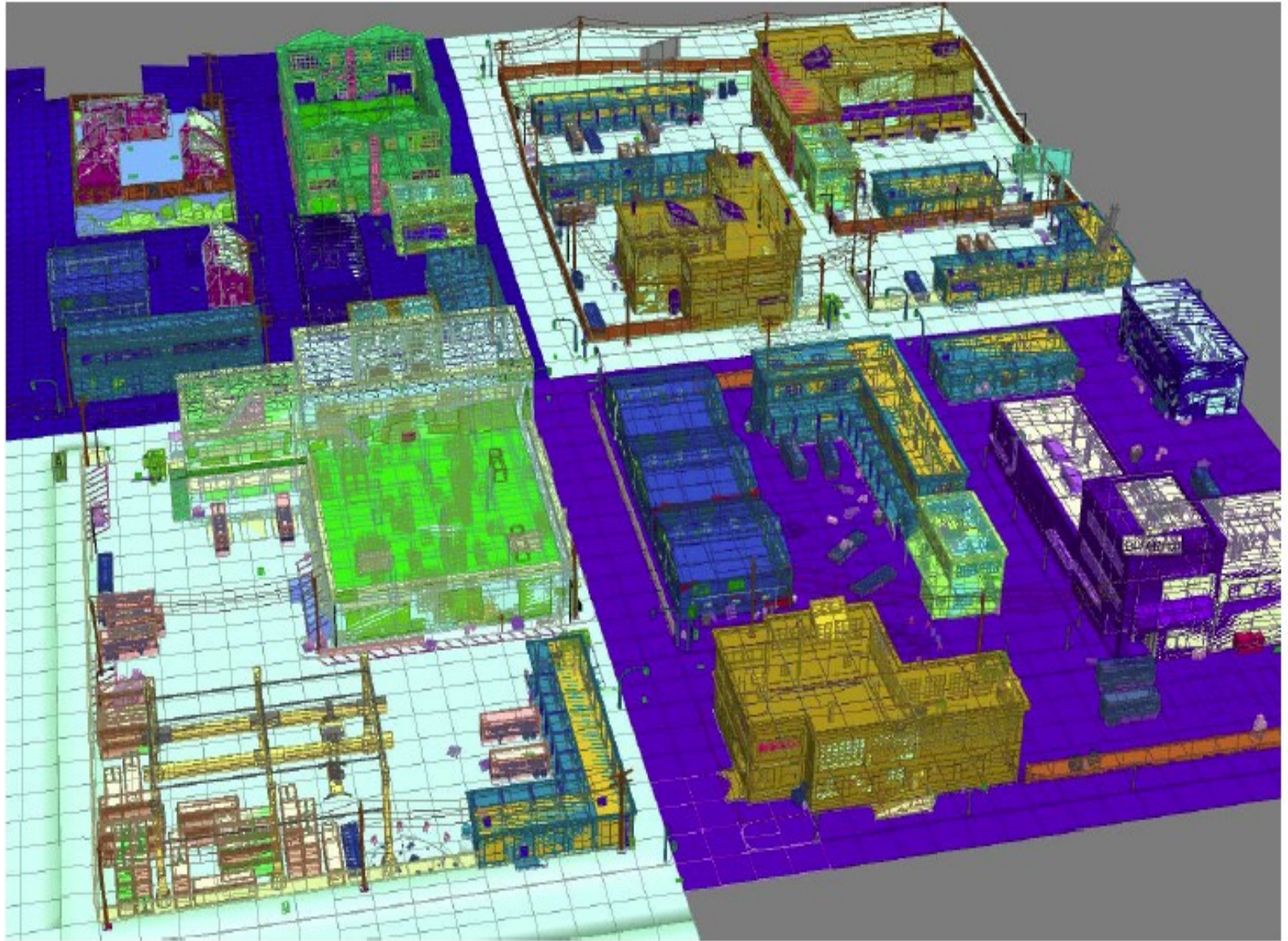


Saints Row는

- **GTA III, Vice City, True Crime 류**의 게임
- 도시 갱들의 싸움
 - 현실적인 밀도 높은 도시
 - 거주자는 뒷뜰을 가지고 있음
 - 다양한 길들과 상점들이 존재
 - 빌딩의 크기는 실제 비율



프로덕션 이전의 프로토타입시절의 레벨





프리프로덕션 끝 / 프로덕션 시작

- 프리프로덕션의 끝
 - 도시 지역 프로토타입이 필요
 - 대략 **4개** 도시 블록 사이즈
 - 도시 제작과 프리프로덕션의 목표가 교차하도록 디자인
 - 엔진에서 반드시 필요한 기반 작업 준비
- 프로덕션 작업을 시작하기 위한 최종 디테일 결정
 - 항공기를 만들 시간은 부족
 - 섬이 아니며 지형적인 경계를 가짐
 - 시카고 / 디트로이트 스타일
 - **GTA Vice City** 정도의 규모

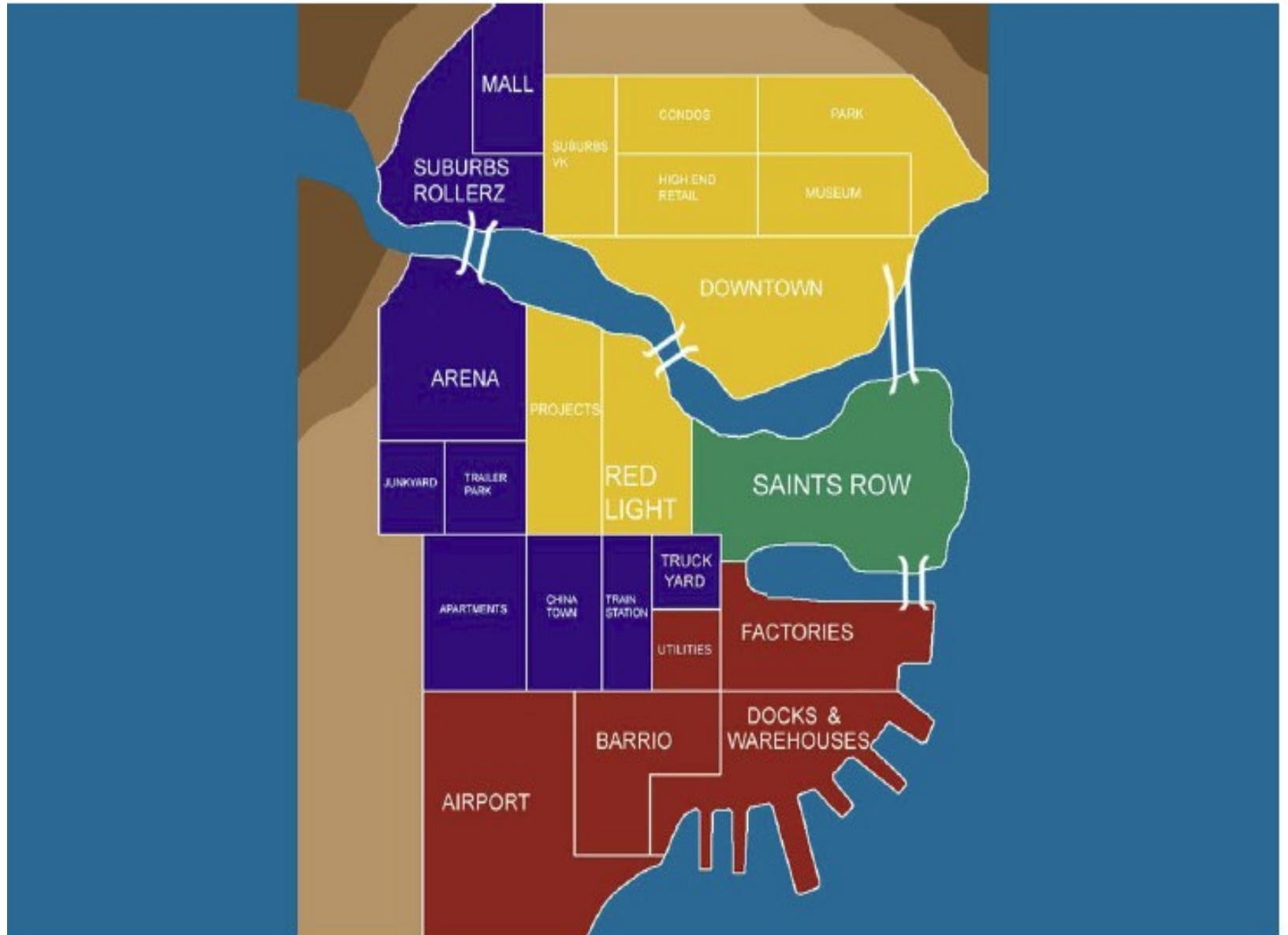


차별화 전략

- 더 많은 게임 플레이 **More Gameplay Per Foot**
- 고르게 분포된 게임 플레이
- 포그 없는 고밀도의 도시 느낌
- **Havok** 물리
- 더 많은 들어갈 수 있는 빌딩
- 더 많은 디테일, 더 많은 장치, 더 특별한 그래픽
- 복사된 듯한 도시 블록



초기맵



디자이너가 처음 만든 월드맵





Stilwater의 고도





월드 제작

- 스테이지 별로 월드 제작

- 스테이지 **1**

- 거리 레이아웃과 큰 건조물

- 스테이지 **2**

- 러프한 모델링과 플랫 셰이딩

- 스테이지 **3**

- 최종 빌딩 모델, 지형, 텍스처링



스테이지 1 제작

- 지역당 한명의 그래픽 디자이너
- ## 2d 컨셉





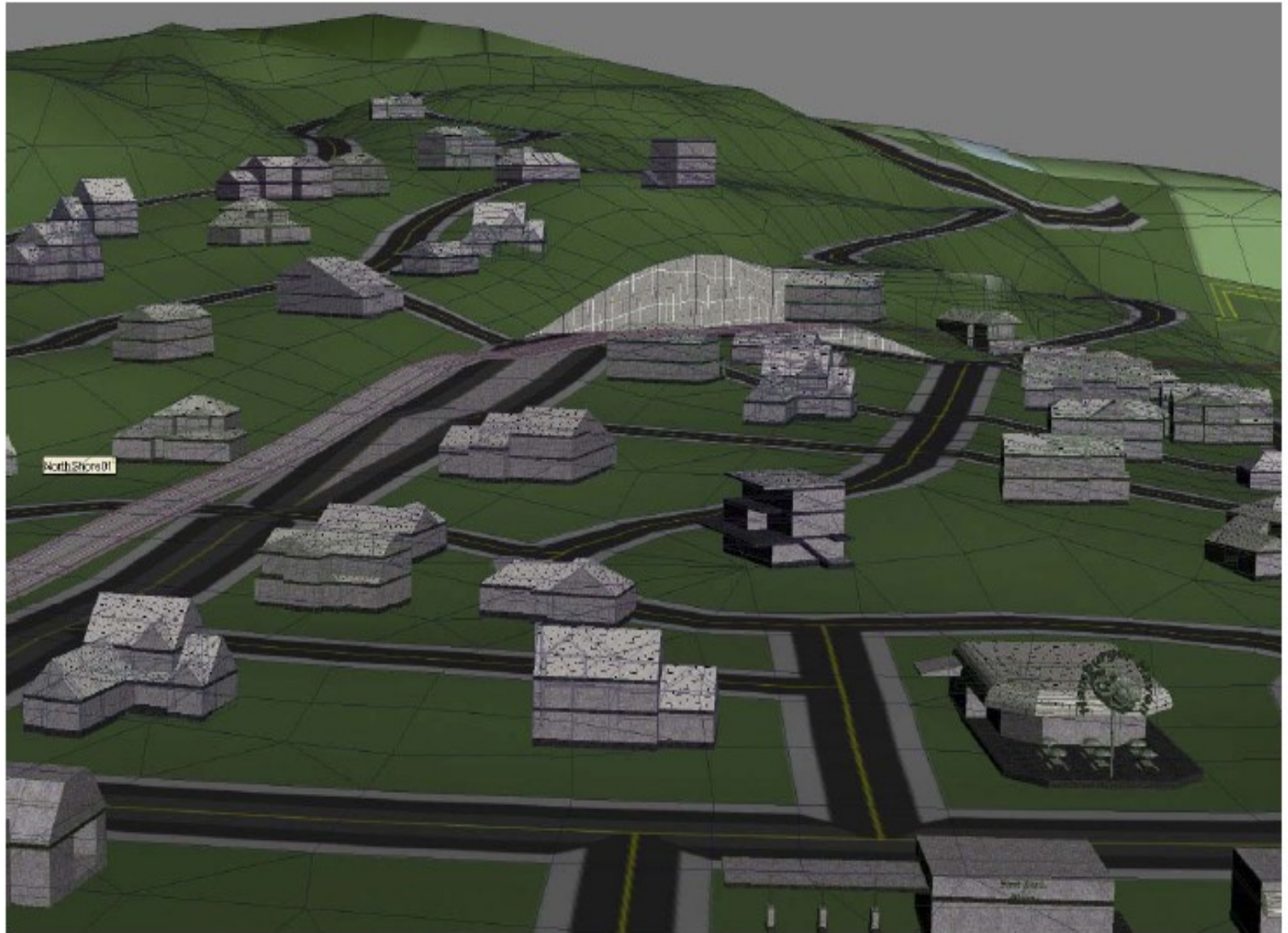
스테이지 1 제작

- 지역당 한명의 그래픽 디자이너
- **2d** 탑뷰로 지역 컨셉 만들기
 - 관심 포인트
 - 건물 배치
 - 보도, 주차장 등
- 한 지역에 평균 **15**개의 건물
- 길은 **spline** 곡선으로 설정
 - 길은 고정된 너이로
 - **spline** 은 인접 지역과 공유





스테이지 1 종료





스트리밍 개요

- 청크 (덩어리, 그룹) 스트리밍
- 내부 스트리밍
- 청크 파이프라인
- 항상 읽어두기
- 메모리



청크 스트리밍

- 스크리밍 시스템의 종류

- 진행형

- 한번에 하나의 오브젝트를 읽어들이
- 문제:
 - 밀도가 높은 장소에서는 제대로 작동 안함
 - **dvd** 처리량보다 **CPU, GPU**의 성능이 훨씬 발달됨
 - **DVD** 검색(이동)시간은 밀도 높은 지역에서 큰 부담

- 청크

- 한번에 많은 데이터 세트를 읽어들이
- 이 방식 선택
 - **DVD** 검색(이동) 시간에 최적화



청크 스트리밍

- 실제 지역 크기 **vs** 메모리
 - 물리적 크기
 - 도시 디자인에 따라 다름
 - 빠른 자동차로 이동 가능한 시간에 맞춰 제한
 - 대략 **4,5** 초정도 청크를 읽는 시간이 걸림
 - 메모리
 - 두개의 청크를 스트리밍
 - 하나의 청크는 **55MB**, 총 **110MB**
- **Note:** 이 시점에서선 실기가 없어서 대략적인 스펙으로 정한 수치임



- 스트림 트리거
 - **disc** 에서 읽기 시작하도록 정의 된 영역이 존재
 - 임의 크기
 - 보통 청크보다 큰 사이즈
 - 항상 렌더되는 영역의 바운드를 따르지 않음
 - 디스크에서 다음 청크를 읽을 시간을 확보해야 함
 - 융통성 있게 할 수록 문제가 생김
 - 항상 도시 디자인과 맞지 않음



내부 스트리밍

- 목표는 내부를 들어갔나 나올때 로딩화면이 뜨지 않는 것
- **8 MB** 메모리풀
- 겪은 몇가지 이슈
 - 상점을 로딩할때 윈도우를 통해 보인다
 - 오브젝트 중 안과 밖에서 보이는 오브젝트를 프래그로 관리함
 - 게임 디자인적인 목적이 아니면 문을 일반적으로 닫아둠
 - 내부끼리 보이는 시야 라인은 피해야함



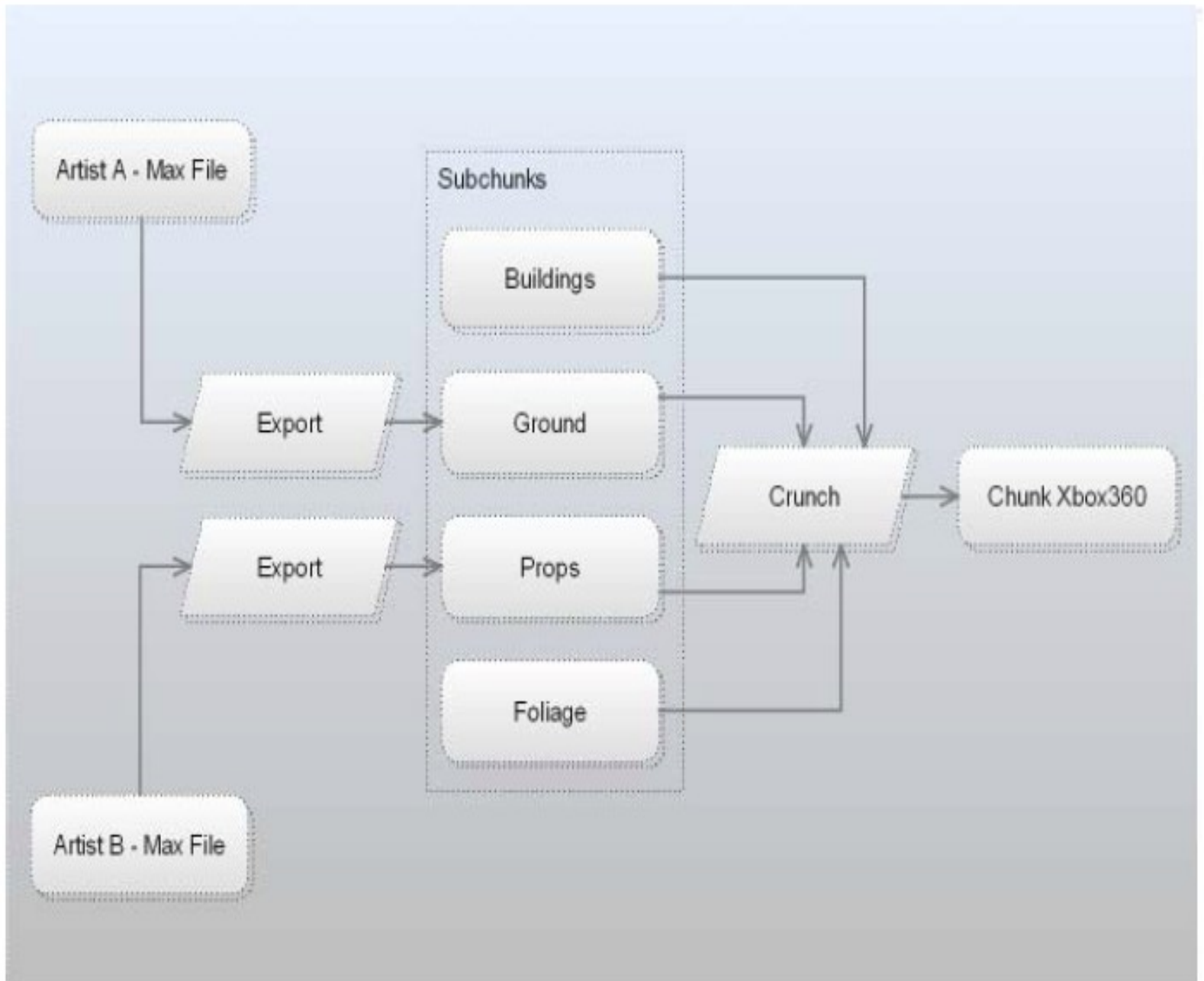
청크 파이프라인

- 게임의 복잡도로 예상하지 못한 문제들이 생김
 - 소수의 디자이너에서 금방 **20명** 넘는 팀으로 커짐
 - **3D Studio Max** 는 필요한 거대한 데이터를 다룰수 없음
 - 우리는 거대한 레벨을 작은 영역으로 나눠서 작업
 - 프로덕션 초기에는 익스포팅 시간만 한시간이 넘게 걸림
 - 다른 익스포트 프로세스를 개발하게 됨



서브 청크 파이프라인

- 청크를 서브 청크라고 불리우는 작은 역역으로 나눠서 작업
 - 서브척트는 일반적으로 아래처럼 나눔
 - 땅
 - 건물
 - 소품
 - 나무, 잎
 - 내부
 - 이펙트
 - 작업 효율을 높이기 위해서 각 서브청크를 동시에 직행
 - 익스포팅 시간이 한시간에서 몇 분으로 줄어듬





항상 미리 읽어둬

- 항상 읽어두는 파트를 명시함
- 목표는 멀리 보이는 지역으로 아주 크고 복잡한 도시에 와있다는 느낌을 주도록
- 본질적으로 고해상도의 도시를 저해상도로 표현한다고 봄
 - 고해상도와 동일하게 분리해서 가지고 있음
- 건물, 특징이 되는 구조물, 땅으로 구성



- 값싼 셰이더로 렌더링
 - 메모리를 절약
 - **GPU** 부하는 적음
- 고해상도의 청크가 로딩되기 전에는 저해상도의 청크로 렌더링
 - 굉장히 튀는 문제가 발생
 - 건물 외곽선은 일치하지 않고, 거의 박스 형태로 가지고 있기 때문
 - 텍스처 해상도와 셰이더도 다름



- 우리가 마주친 문제들을
 - 플레이어들이 가끔 스트리밍 트리거의 경계에서 항상 읽어두는 저 해상도의 지형들을 볼 경우가 생김. 그래서 두가지 방법으로 접근
 - 스트리밍 예측 - 플레이어의 속도와 방향을 이용해서 예상. 다음 지역으로 가기 몇 초전에 스트리밍을 시작하게 됨
 - 오브젝트 프래그 - 프래그를 두어서 만약 고해상도 청크와 저해상도 청크가 읽히면, 그 오브젝트를 그대로 렌더링



메모리

- 모든 콘텐츠는 적어도 두 개 이상의 파일로 구성되어 있다.
 - 하나는 메시 데이터, 다른 하나는 텍스처 데이터. 메시 정보에는 충돌 정보들을 포함하기 때문에 항상 먼저 읽고, 텍스처 데이터는 그 후에 읽는다



청크 메모리

- 모든 청크는 4개의 파일로 구성되어 있다
 - **Chunk** - 모든 메시 데이터
 - **Peg** - 모든 텍스처 데이터
 - **Hmap** - AI 헬기를 위한 높이맵 정보
 - **PVS** - 미리 계산된 **Visibility** 정보

Level Mempools	Size
Chunk	55 MB * 2
Interior	8 MB
Always Loaded	76 MB
Misc Permanently Loaded Textures	11 MB
Total	205 MB



- **Xbox 360 메모리 패딩제한이 문제가 됨**
 - 이 규칙은 mip맵이 있는 텍스처와 **128x128** 보다 작은 텍스처는 자동으로 공간에 맞게 패딩됨
 - 프로덕션 초기에 하드웨어 스펙만으론 예상하지 못 한것
 - 때문에 **Pegs**(텍스처) 데이터들이 컨트롤 안 되거나 메모리에 딱 맞지 않게됨
 - 'mip맵 인터리빙'이란 방법으로 구현
 - **Peg** 파싱에 낭비되는 공간을 분석해서 메모리에 딱 맞도록 재 구성함
 - 이 작업으로, 텍스처는 이 룰을 준수해야 함



다름 메모리 이슈

- 일반적으로 셰이더는 **3**종류의 텍스처를 사용함
 - **Diffuse Map**
 - **Normal Map**
 - **Specular Map**
- 만약 셰이더가 하나 이상의 **UV**채널을 사용한다면 각 **UV** 채널의 메모리 비용이 더 소비됨
- 메모리를 절약하기 위해, 텍스처를 **AlwaysLoaded_User peg** 라고 불리는 메모리풀에 넣고 사용함

건물





- 건물은 외주로 제작

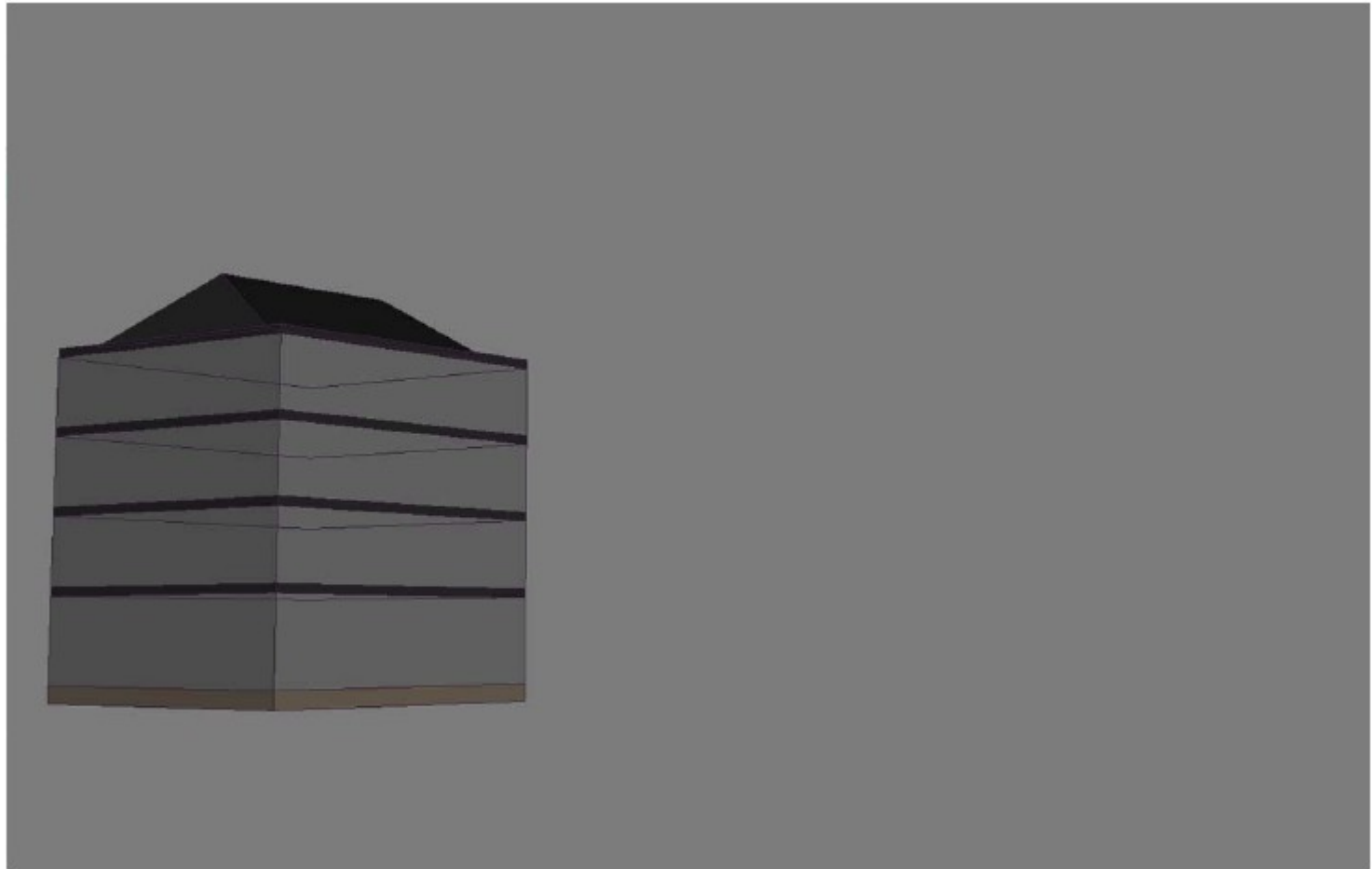
- 4개의 카테고리

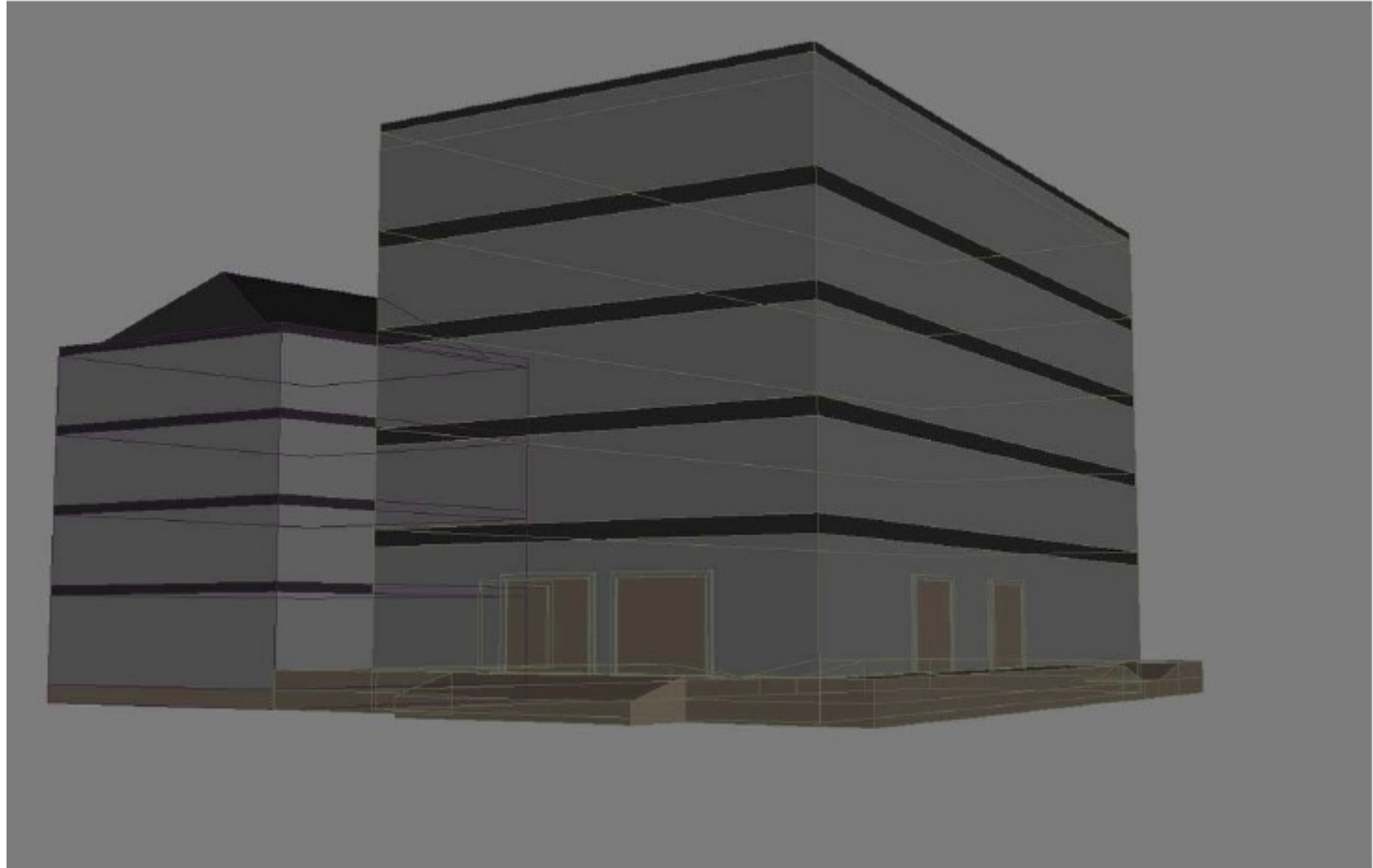
- 특징이 되는 건물
 - 들어갈 수 있는 상점
 - 들어갈 수 있는 본거지
 - 일반적으로 매꿀만한 건물

- 카테고리에는 복잡도로 정렬

- 복잡도는 작업 시간으로 측정
 - 기준 단위는 일주일 작업치
 - 외주로 맡기기전에 러프하게 모델링을 해둬

Three Proxy Detail Levels









빌딩관련

- 더 세밀한 빌딩은 더 작업을 해야 함
 - 최종 모양이 더 쉽게 전달됨
 - 더 정확한 배치를 허용
 - 건물의 출입문은 게임플레이하면서 작업
- 최종 건물 스케줄
 - **+ -300** 단위 작업시간이 필요한
 - **6 Man Years**
 - **100** 개가 넘는 건물
 - **10달**의 프로덕션 시간이 남았고, 다 만들 시간이 부족함
 - 가능한 모든 건물을 외주로 제작

Example of Shader Features





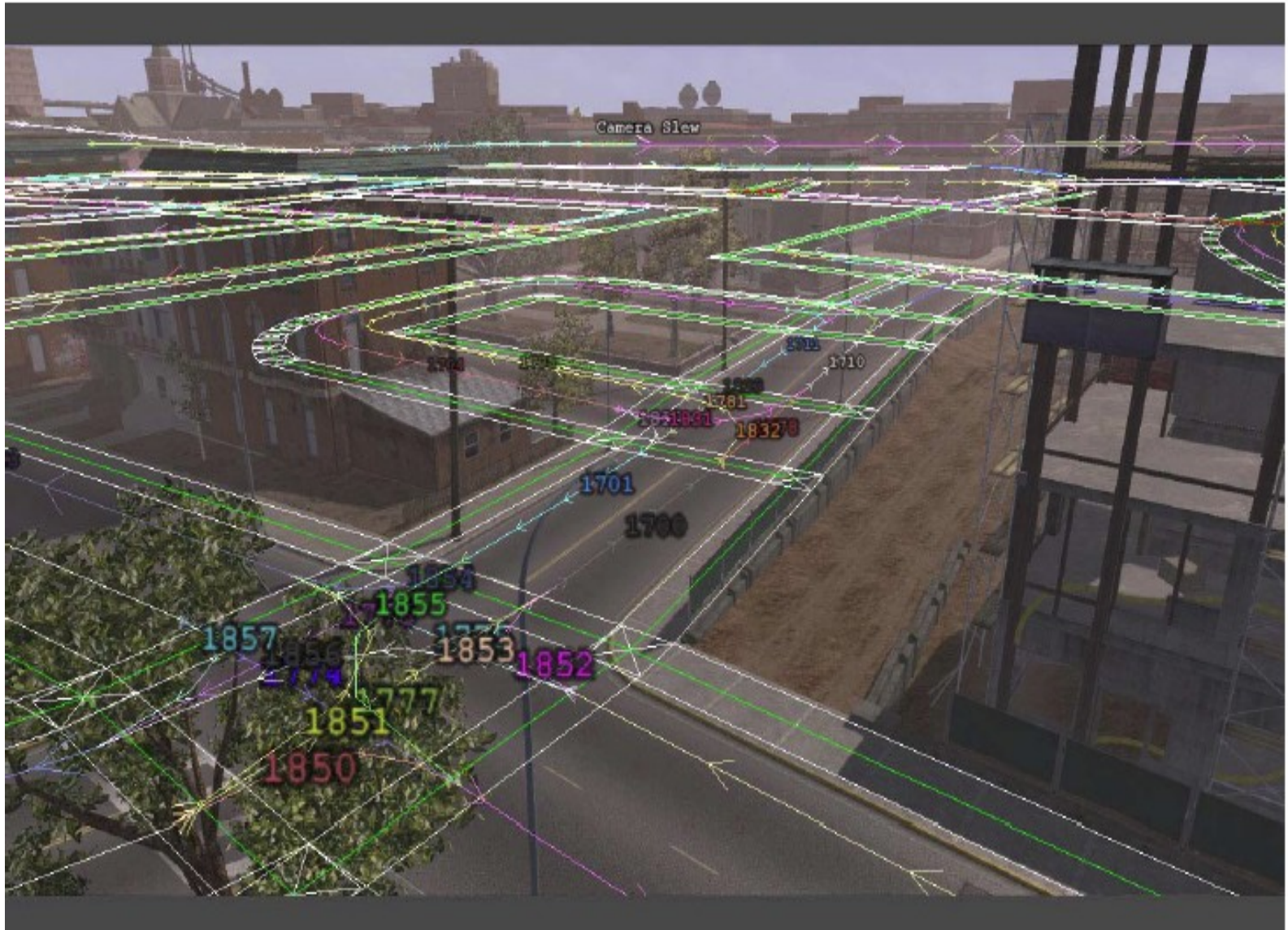
스테이지2

- 스테이지2의 목표

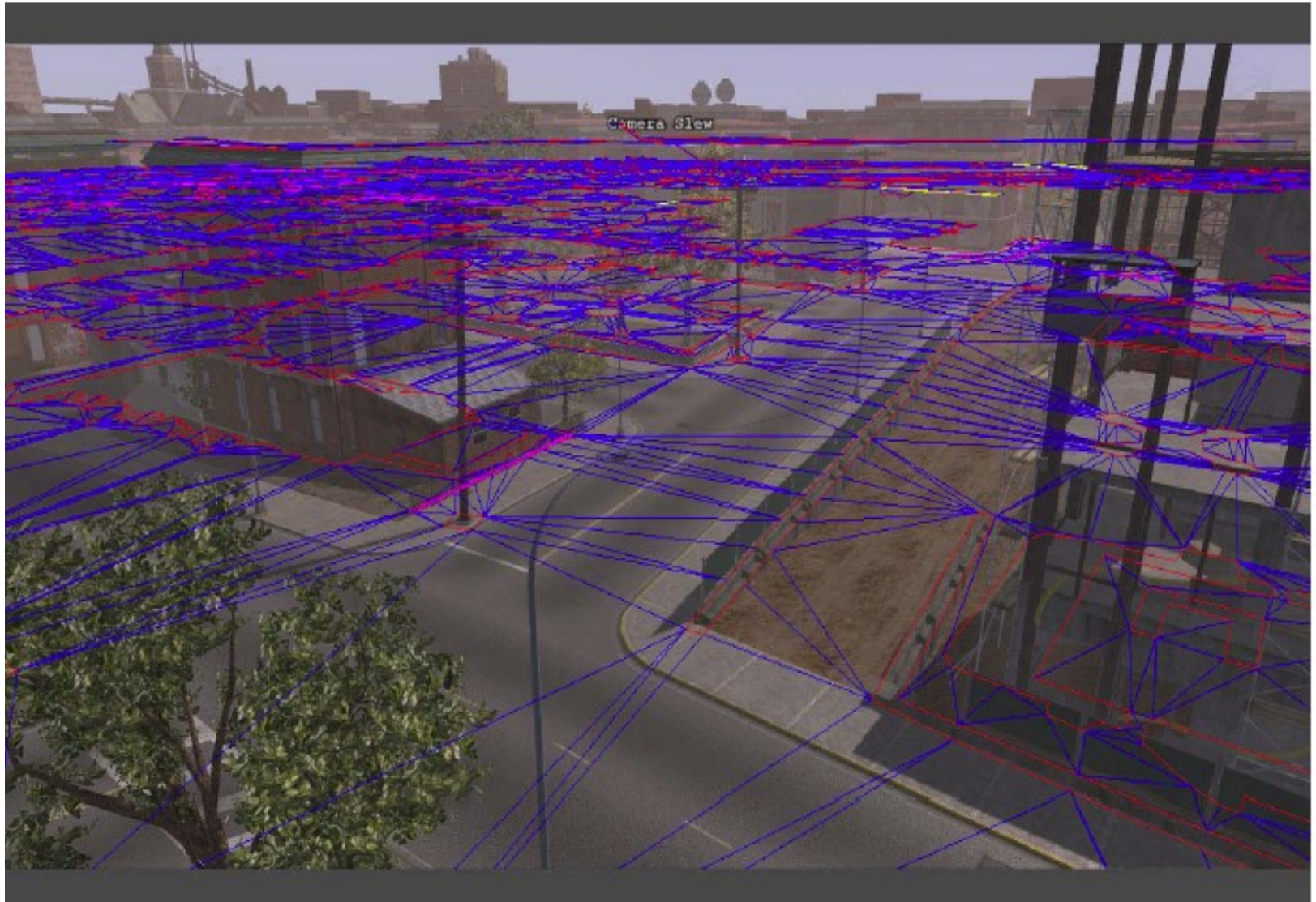
- 게임의 플레이가 가능하게 잘 동작
- 지역의 테마가 잘 표현되어 전달됨



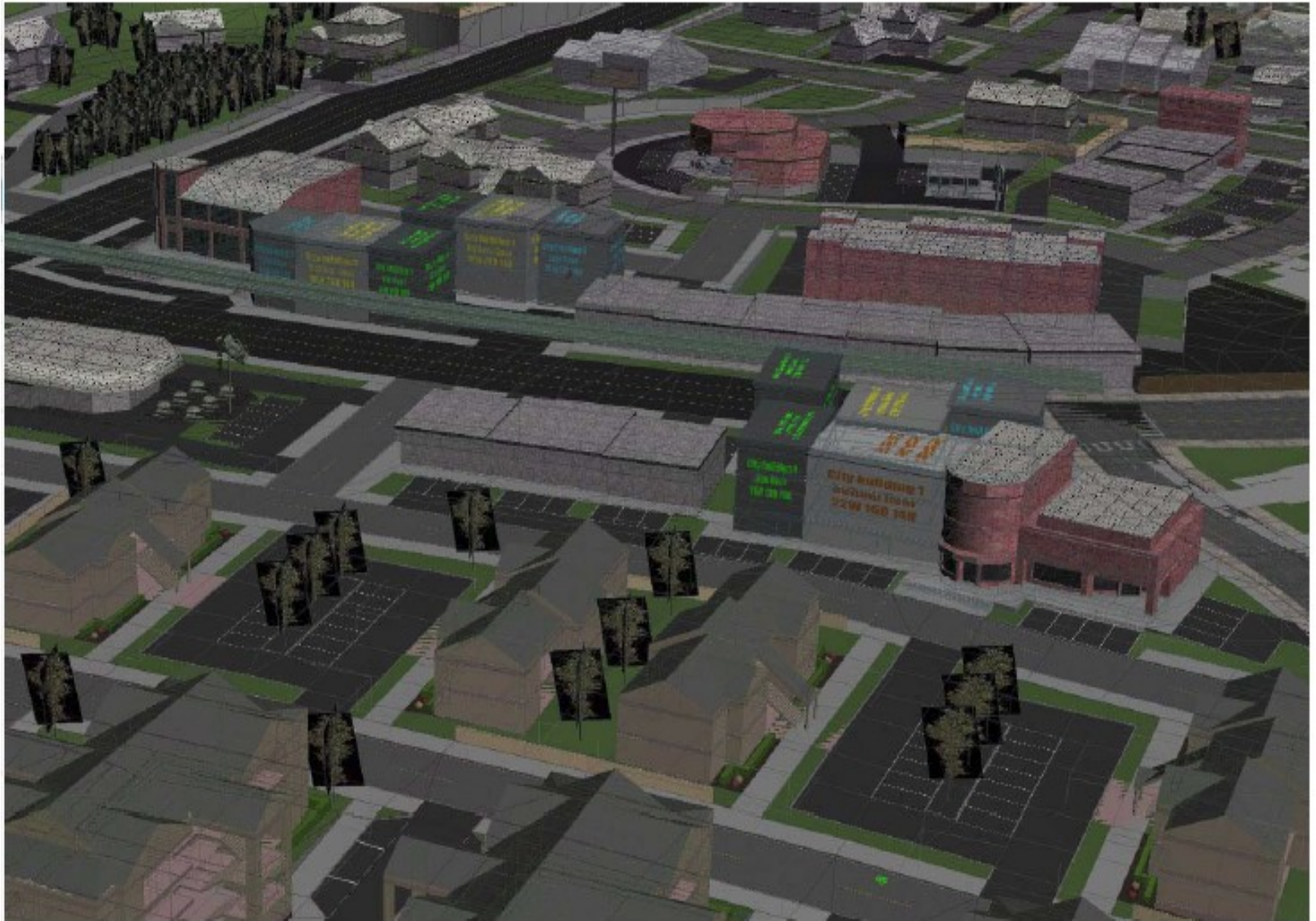
네비게이션 **splines**



World 네비게이션 splines



스태이지2 결과







E3

• E3 데모

- 스트리밍은 작업하기가 어려움
- 기대한 것보다 완성하는 데 더 오래 걸린다고 판단
 - 데모 작업을 위해 더 많은 팀원을 투입
- 레벨에 사용되는 메모리가 제한된 메모리보다 큼
 - **150%**



E3 해결책

- **E3** 데모를 메모리에 넣을 수 있도록 관리
 - **E3**전에 **MS**는 **360**의 메모리를 두배로 키워줌
 - 여전이 **60**메가를 초과
 - 거대한 메모리 풀을 사용하는 스트리밍 데모를 제거
 - 고유한 건물을 절반으로 줄임
 - **13**개까지 줄임
 - 계획된 건물 수정 작업을 하지 않음
 - 가능한 다른 메모리풀의 여유 메모리를 끌어다 사용

Game Developers
Conference 07



TAKE CONTROL
March 5-9, 2007 in
San Francisco



CMP

Computer Music Products

- Saints Row vs GTA:SA





E3 후

- **E3 데모를 완료**

- 현실적인 스케줄이 필요해짐
- 데모에서 메모리는 예산을 넘어섬
- 스트리밍 확인한 테스트가 필요함
- 프레임 레이트도 출시 수준은 안됨



현실적인 스케줄 짜기

- 모든 디자이너가 월드를 평가
 - 각 블럭을 리뷰
 - **400** 블럭
 - **3** 일
- 블럭을 주어진 스케줄로 평가
 - 현재 **5년 6월**
 - **1월**까지 콘텐츠 완성
 - 도시 스케줄을 거의 **20 Man Years**의 작업량
 - 스케줄 \circ **7년 4월**까지로 늘림



도시 완성 계획

- 유니크 아트의 양을 다시 돌림
- 도시 사이즈를 가능한 크기로 줄임
- 외주로 더 할애
 - 내부
 - 내부 디자이너를 외부 제작으로 할당
 - **16 of 20** 의 우선 과제들
- 월드를 평평하게 함



Saints Row는

- **GTA III, Vice City, True Crime** 류의 게임
- 도시 갱들의 싸움
 - 현실적인 밀도 높은 도시
 - 거주자는 뒷뜰을 가지고 있음
 - 다양한 길들과 상점들이 존재
 - 빌딩의 크기는 실제 비율



Saints Row는

- **GTA III, Vice City, True Crime** 류의 게임
- 도시 갱들의 싸움
 - 현실적인 밀도 높은 도시
 - 거주자는 뒷뜰을 가지고 있음
 - 다양한 길들과 상점들이 존재
 - 빌딩의 크기는 실제 비율



마지막 푸시

- 최종 허들은 월드 끝내는 것
 - 청크를 메모리에 넣기
 - 스트리밍 작업
 - 더 빠르게
- 메모리 이슈
 - 많은 특별한 디자인(**unique art**)
 - 청크별 특별한 디자인 제거
 - 청크별 **20-30**개의 특징있는 건물이 있음
 - **12-13**로 축소
 - 텍스처를 줄이고, 통합, 등등
 - 도저히 안되는 청크는 절반으로 나눔
 - 스트리밍 생성 문제



- 스트리밍 이슈
 - 플레이어가 청크를 읽기전에 도착
 - 장애물을 배치해서 속도를 늦추게함
 - 길을 재 정비해서 막음
 - 재작업이 발생함
- 마지막 과제
 - 프레임 레이트



퍼포먼스

- 셰이더
- 셰이더 **LOD**
- **Draw** 함수 호출
- 오클루전
- **PVS (Pre-Computed Visibility)**



셰이더

- 아트 파이프 라인에 셰이더를 사용하는 것은 처음
- 테크니컬 아티스트가 셰이더 개발. 랜더링 프로그래머가 가능한 최적으로 최적화
- 실기가 없는 관계로 알파 하드웨어와 **PC**에서 개발. 그래서 최적화 정도가 불분명함



- 셰이더

- 필레이트가 낮에는 문제가 안되는데, 밤에는 헤드라이트와 길거리 조명들로 문제가 됨. 중요한 문제
- 프레임 레이트가 시점에 따라 달라지지만, 일반적으로 다음 요소를 감안
 - 셰이더 요소 (i.e. 비싼 **ALU**, 텍스처 룩업 테이블의 수)
 - **Draw** 함수 호출 수
 - 셰이더의 종류
 - **CPU**는 새로운 셰이더를 **GPU**에 올리는 비용 때문



셰이더 LOD

- 어느 정도의 거리의 모델에 할수 있는 것은 셰이더의 기능을 효과적으로 끄는 것
 - 밤시간의 필레이트 확도에 도움이 됨
 - 다음에 의해 효과가 있음
 - 각 셰이더는 개별 lod규칙 파일을 가지고 있으며, 이 규칙 파일에 따라 기능을 활성화하도록 명령할 수 있음.
 - 길시간으로 셰이더 기능을 끄는 방법이 없기 때문에 셰이더 담당자는 룰에 따른 **LOD** 버전을 만들어야 했음
 - **30미터** 마다 적당한 **LOD** 버전으로 교체



전형적인 셰이더의 **LOD** 사용 예

LOD 1 – 30 meters

Normal Map (this has the least amount of visual impact)

Normal Map Lighting from the Vertex/Pixel Shader

LOD 2 – 60 meters

Specular Map

Specular Lighting from the Vertex/Pixel Shader

LOD 3 – 90 meters

Decal Map 1

Decal Map 2



Draw 함수 호출

- 오브젝트의 밀도 때문에, 커맨드 버퍼는 대개 꽉차서 넘침
- 주어진 프레임에 적당한 평균 오브젝트 수는 쉽게 **1000개** 정도
 - 프로덕션 타임에 **PVS (Pre-Computed Visibility)**를 개발할 수 밖에 없게 됨
 - **PVS**로 보통 **800-1000** 가 됨



오클루전

- **PVS**사용하기 전에, 오클루더를 오클루전 용도로 먼저 적용함
 - 오클루더는 박스나 이등변의 삼각형으로 구성
 - 대부분의 건물, 별, 대형 오브젝트에 설정
 - 이런 종류의 오클루전은 선형적인 레벨 디자인에 적합함
 - 시야가 넓은 거리나, 해변, 강가등 가리는 것들이 별로 없는 곳에는 불리함



- 하이브리드 오클루전 시스템을 개발
 - 오클루더는 **PVS**로 걸러내지 못하는 경우를 대비해서 그래도 남겨둬
 - 동적인 오브젝트에 대해서 여전히 사용
 - 자동차
 - 보행자
 - 나무
 - 소품들 (동적인 가로등, 벤치, 등.)

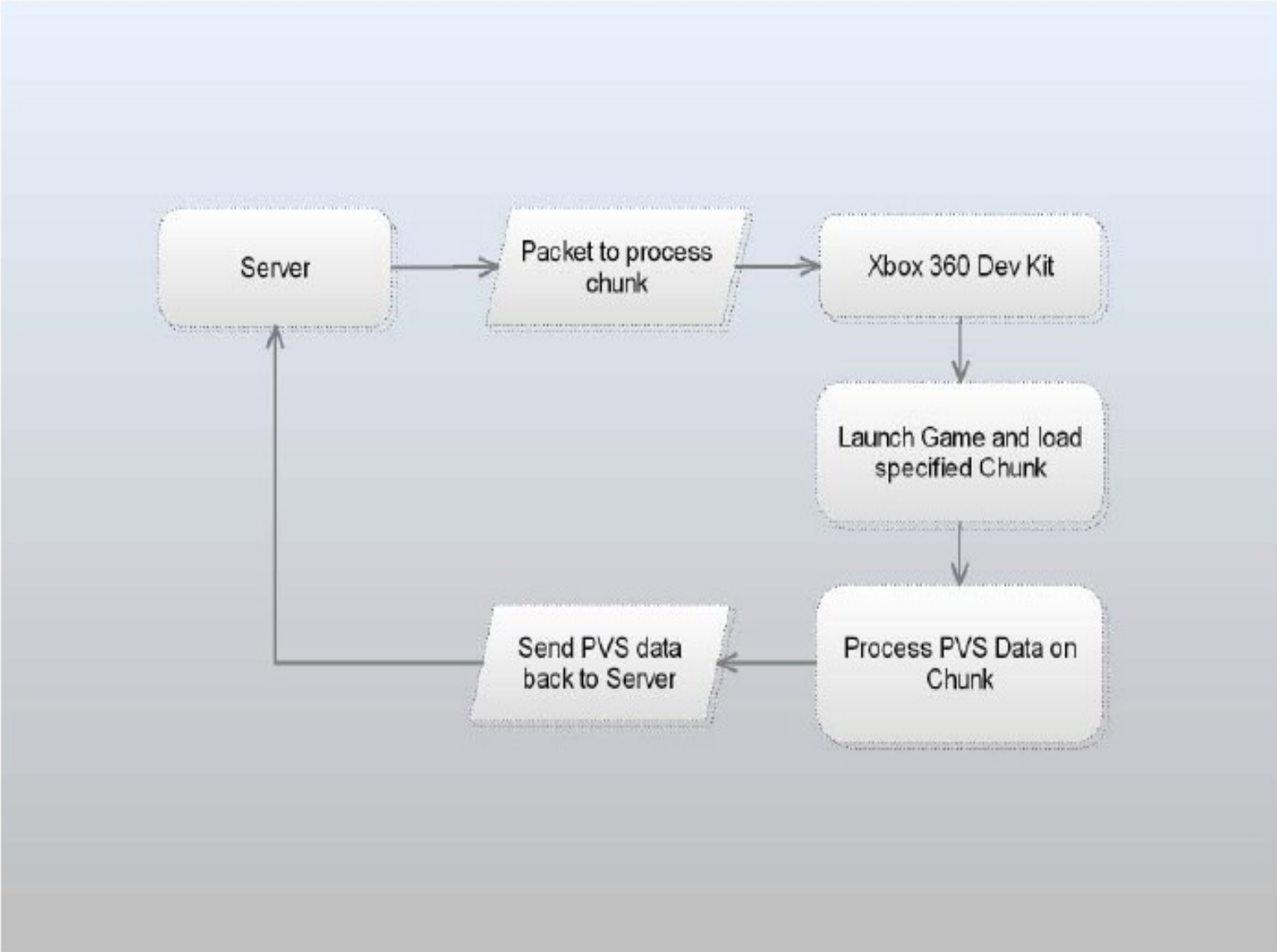


Pre-Computed Visibility (PVS)

- 정적인 지오메트리에 대해서 가림 처리
- 출시를 위해 빠르게 **2달**만에 완성
 - 한달은 개발
 - 한달은 버그 수정
- 우리의 마지막 궁지는 일반적인 상황에서 **30프레임**으로 게임이 돌아가는 것



- **PVS**계산은 **Xbox 360** 개발 팜에서 자동으로 계산. 단계는
 - **DevKit** 에 청크 계산과 관련된 다양한 패킷을 보냄
 - **DevKit** 에서 패킷을 받으면 게임을 실행하고 **PCS**데이터를 생성함
 - 청크는 그리드 단위로 나뉨
 - 각 그리드 별로 미리 정해진 포인트에 카메라를 배치해서 샘플링함
 - 각 점에서, **PVS** 시스템은 **360** 도 픽셀 레벨로 보이는 지 검사함. 한 픽셀이라도 보인다면 렌더링 프래그를 컴
 - 해당 셀에서의 렌더 오브젝트 리스트가 결과
 - 끝나면 서버에 해당 정보를 보내서 취함





- **PVS**를 돌리는 가장 큰 이슈는:
 - "**object drop out**"이라 불리는 현상이 빈번하게 생김
 - **PVS** 그리드에 들어가면 오브젝트는 갑자기 사라지는 형상이 생김
 - 이를 수정하기 위해 "**spot fixes**" 를 수행
 - 이것은 게임 그리드에 **PVS** 데이터를 생성해서 완성되도록 함. 정보는 디스크에 저장되고 청크 **PVS** 데이터로 통합됨
- **PVS**는 안정적으로 **30FPS** 도는데 도움이 됨