
MARCH 5-9
2007
SAN FRANCISCO
MOSCONE CENTER
TAKE CONTROL
www.gdconf.com
CMP

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

**Presented By**

Kenny Thompson
**Lead Environment Artist**

Jason Hayes
**Senior Technical Artist**

# Concepts

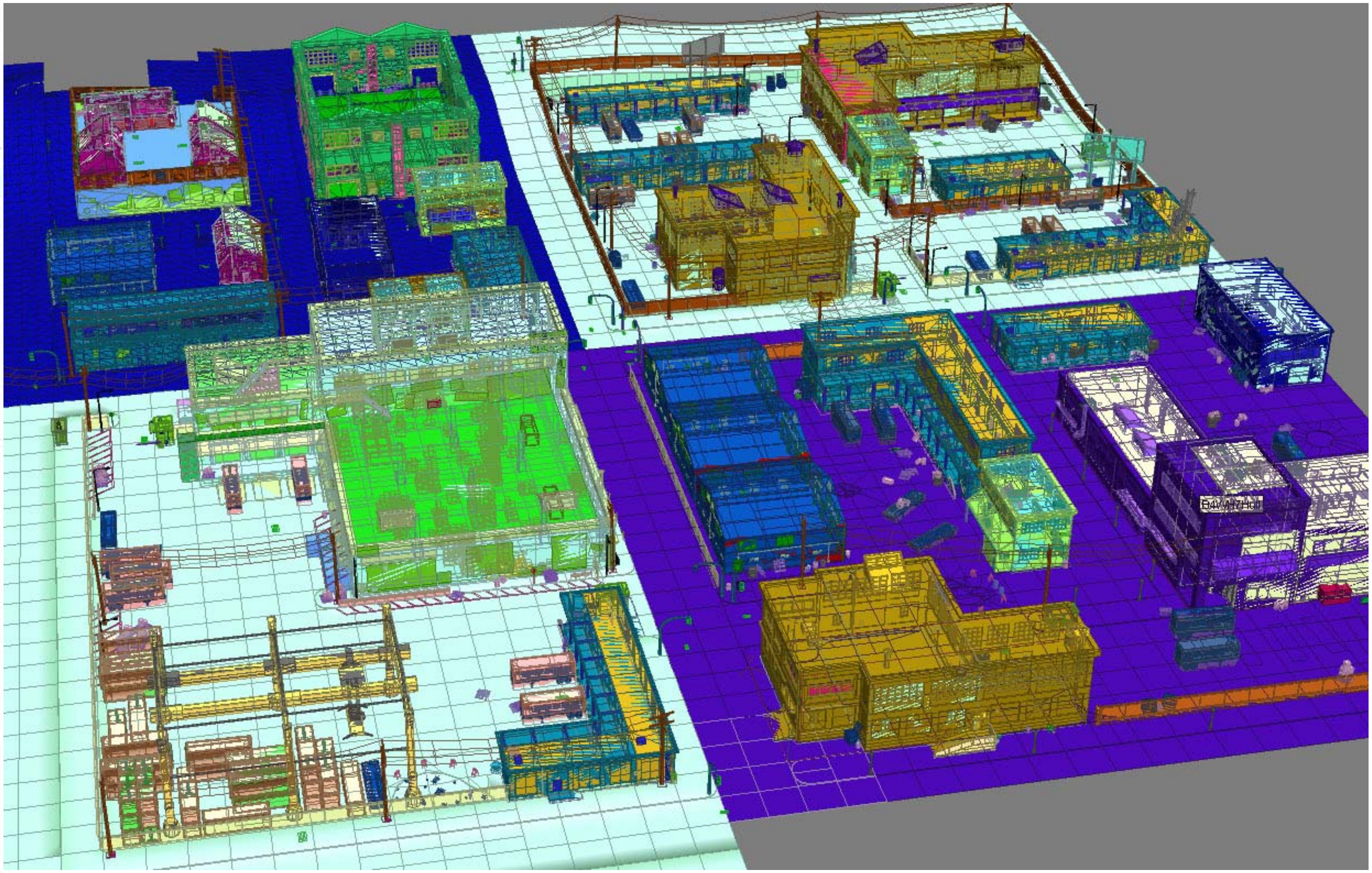
- Stilwater’s Creation Process
  - Production Process
  - Tools
  - Technical Aspects

- New Challenges
  - Volitions first foray into the open world genre
  - No relevant team experience in this genre
  - To be built on a new console with a new engine

## Saints Row’s Premise

- **Similar to GTA III, Vice City or True Crime**
- **Game Design Would be Built Around Gang Banging in an Urban Setting**
  - **Realistic setting in an extremely dense inner city**
    - Residents would have back Yards
    - Block shops with accessible alleys
    - Building Scales would be reasonably accurate

# The Prototype level before production began



# Technical Overview

- Streaming
  - Chunk Streaming
  - Interior Streaming
  - Chunk Pipeline
  - Always Loaded
  - Memory
- Framerate & Performance
  - Shaders
  - Shader LOD
  - Occlusion & PVS

# End of Preproduction / Beginning of Production

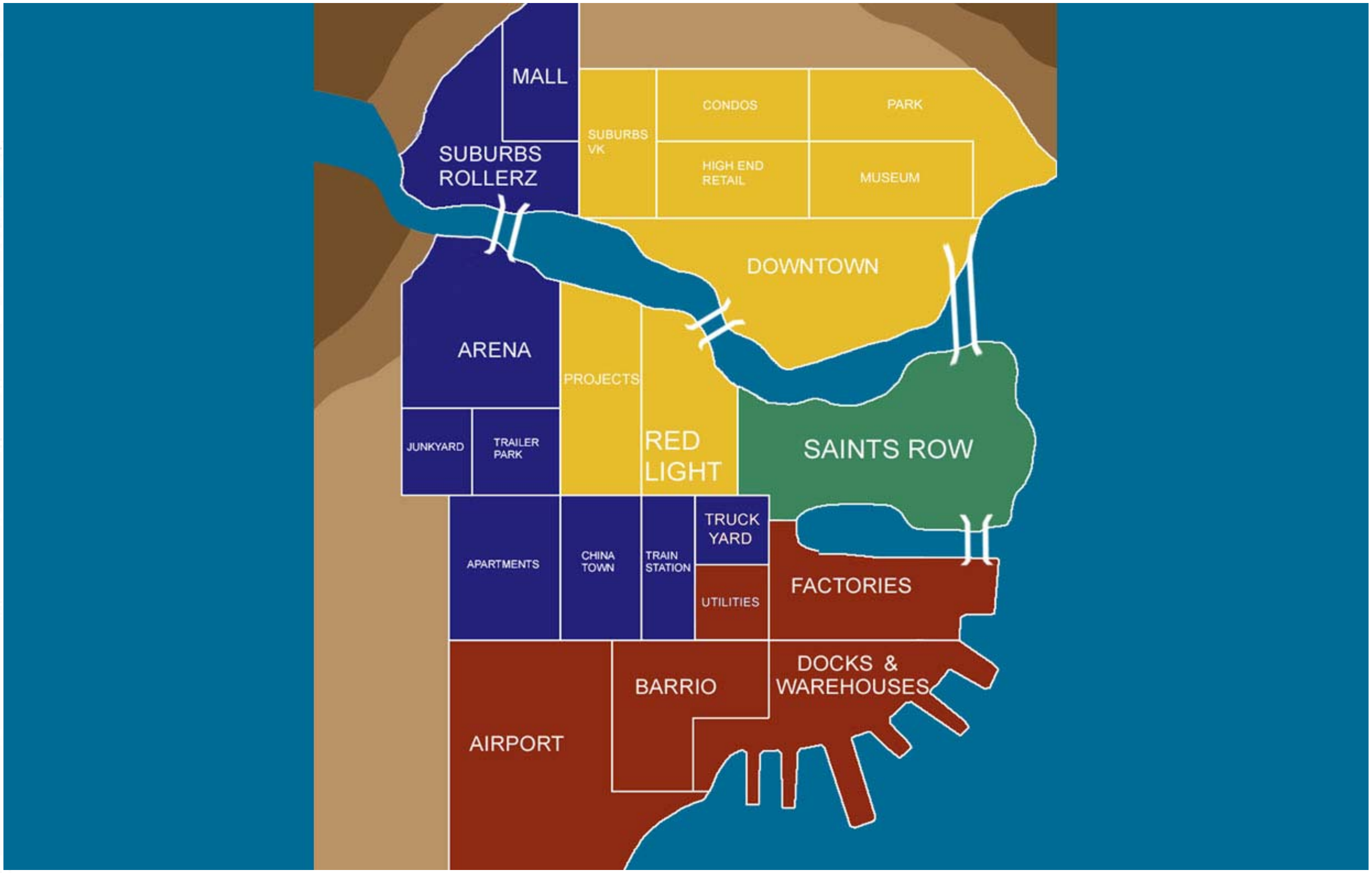
- End of Production
  - Required a Prototype City Section
    - Roughly 4 city blocks in size
    - Designed to meet city construction and preproduction goals
    - Would Provide a proving ground for fundamental engine necessities

- With Last Details Decided Upon Production Began
  - No time for aircraft
  - Stilwater would not be an island world
  - Stilwater would use geographical boundaries instead
  - Chicago / Detroit Style
  - Size of GTA Vice City

# The Plan to Differentiate Our World

- More Gameplay Per Foot
- Evenly Distributed Game Play
- Dense Inner City Feel with No Fog
- Havok Physics
- More Enterable Building Per Foot
- More Detail, More Props, More Unique Art
- No Duplication of City Blocks

# Initial Map of Stilwater



# First Artist Rendered World Map

# Stilwater's Elevation

# World Construction

- **World Construction Took Place in Stages**
  - **Stage One**
    - Street Layout and Major Landmarks
  - **Stage Two**
    - Rough Model and Flat Shading
  - **Stage Three**
    - Final Building Assets, Final Geometry, and Final Texturing

## Stage One Construction

- **One Artist Per Neighborhood**

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

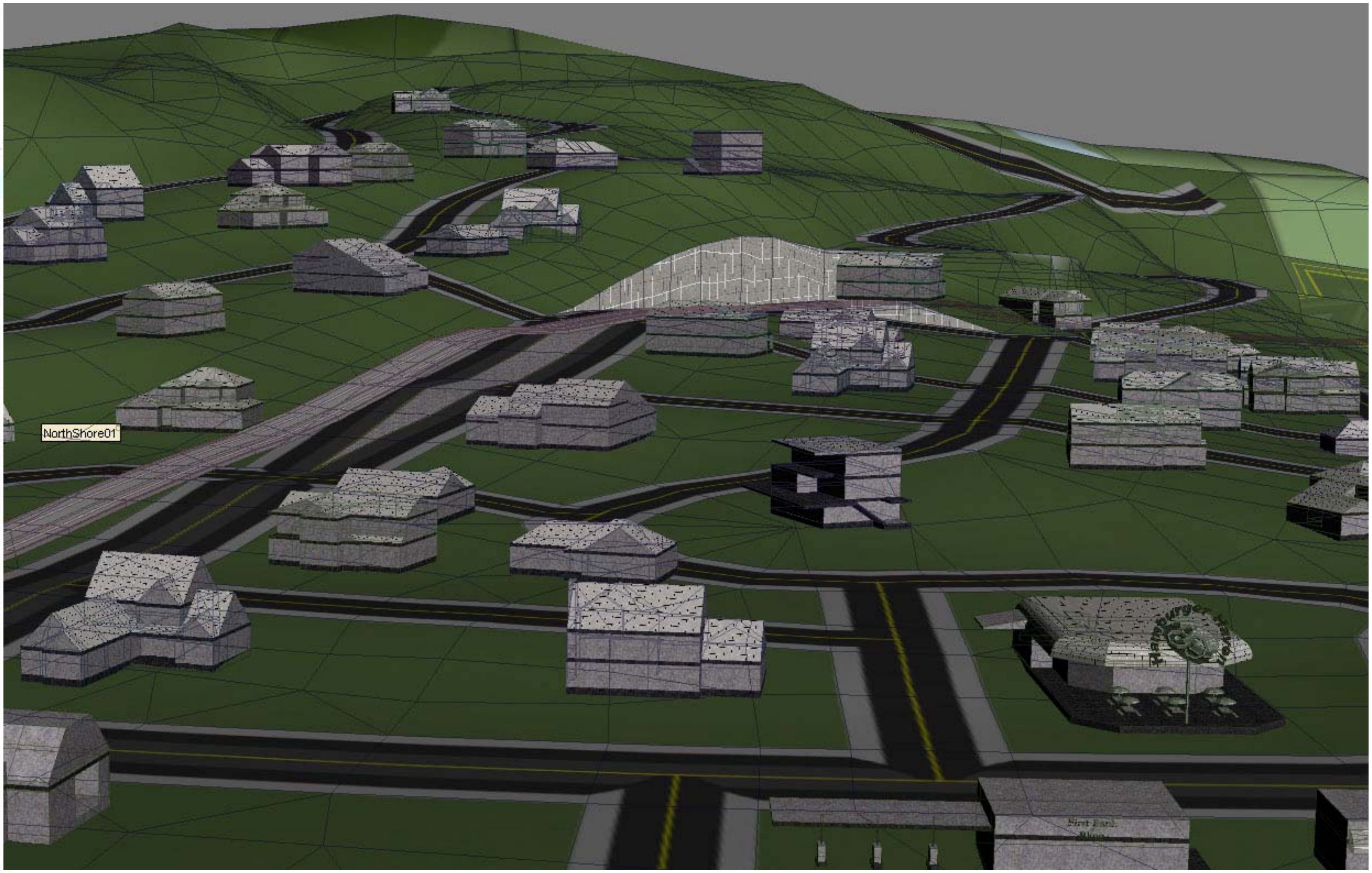
## 2d Concept

## Stage One Construction

- One Artist Per Neighborhood
- Neighborhood Concepted as a 2D Top Down
  - Points of Interest
  - Building Placement
  - Sidewalks, Parking Lots, Etc.
- Average of 15 Building Per Neighborhood
- Road Initially Created as Splines
  - Ensured Roads Had a Fixed Dimension
  - Splines Where Shared with Adjacent Neighborhoods

# Projected Texture with Cut Roads

# End of Stage One



## Streaming Overview

- Chunk Streaming
- Interior Streaming
- Chunk Pipeline
- Always Loaded
- Memory

# Chunk Streaming

- **What type of streaming system?**
  - **Progressive**
    - Stream in objects one at a time
    - Problems:
      - This type of streaming does not work very well in a dense environment.
      - CPU and GPU power have advanced greatly from last generation while DVD throughput has not.
      - DVD seek times were a big risk with our high object density's.
  - **Chunk**
    - Stream in large data set of objects all at once
    - Chose this system because:
      - Addressed the DVD seek times

# Chunk Streaming

- **Real Estate Size vs. Memory**
  - **Physical Size**
    - Subjective to design of the city
    - Limited to time it took to traverse the area in the fastest vehicle
      - Approximately 4 – 5 seconds from one end of the chunk to the other
  - **Memory**
    - Two Streaming Chunks
      - Each chunk would use 55 MB, for a total of 110 MB
- Note: *At this point in development, we were designing our streaming system on specs of the hardware and DVD emulation software provided by Microsoft because there was no hardware to give to developers.*

# Chunk Streaming

- **Stream Triggers**
  - **These are meshes that tell the streaming system what chunk to start loading off disc.**
    - Arbitrary in size.
    - Generally much larger than a chunk.
    - Often did not follow the same chunk border as their respective rendered geometry.
    - Needed to allow the streaming system enough time to load in the next chunk off disc.
  - **While the system was flexible, it had its problems.**
    - Did not always fit well with the design of the city.
    - Stream triggers overlap causing tri-points.
      - These areas became off-limits to the player through content.

# Interior Streaming

- **Goal was to be able to walk in and out of an interior without a load screen. It needed to be seamless.**
- **8 MB mempool**
- **Some issues we ran into were:**
  - **Store fronts that had windows would cause pops when they streamed in.**
    - To get around this, we developed a system to flag certain objects to be visible from the exterior and interior.
  - **Typically kept doors shut unless absolutely needed for gameplay or design reasons.**
  - **Needed to avoid lines of sight between interiors.**

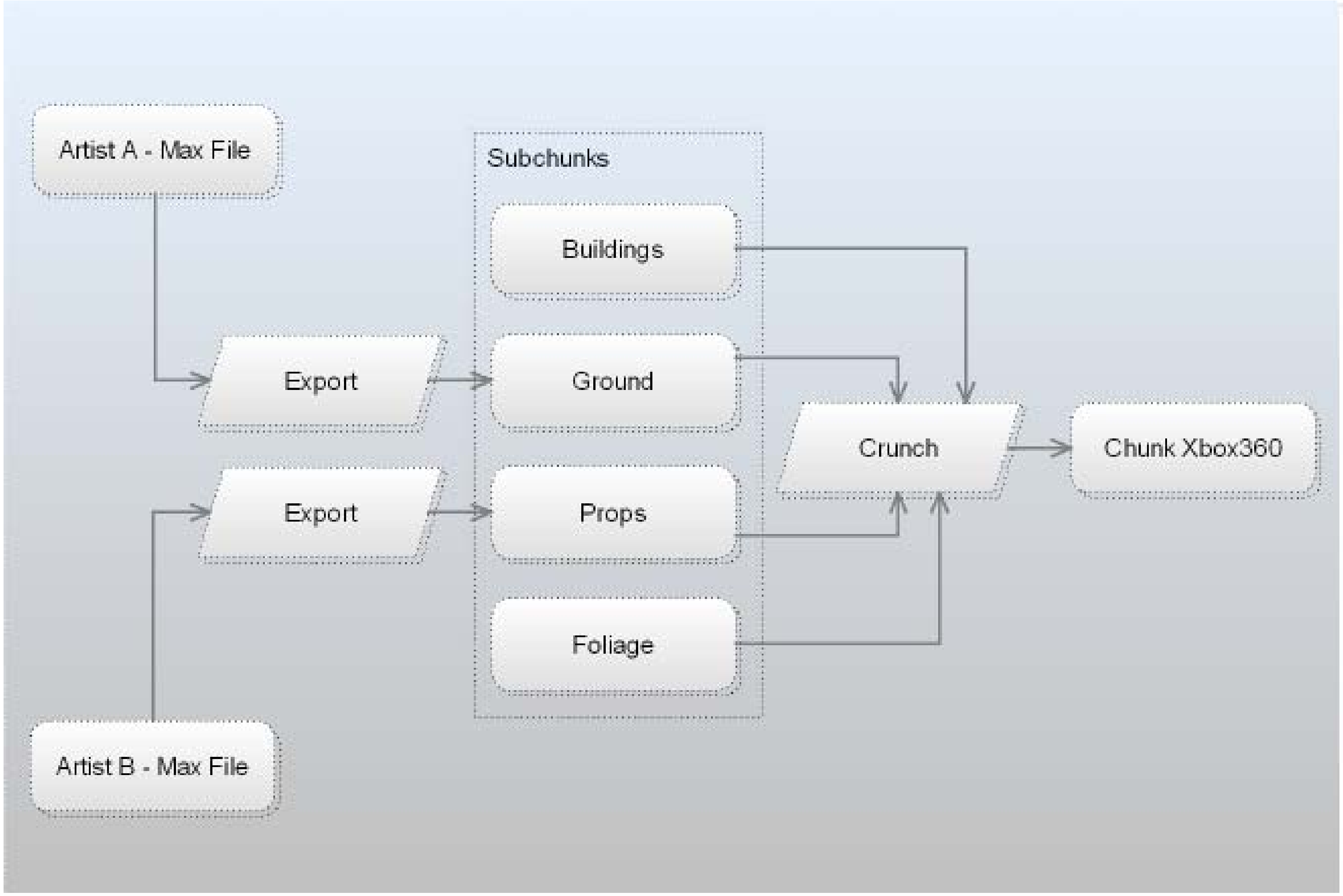
# Chunk Pipeline

- **The complexity of the game presented unforeseen problems:**
  - **Team size grew from a handful of artists to well over 20+ quickly, heavily in environment art.**
  - **3D Studio Max could not handle the large data sets we had.**
    - We are essentially building one gigantic level split up into sections.
  - **Early in Production, exporting time took well over an hour.**
    - These points forced us to develop an alternative exporting process.

# Sub-Chunk Pipeline

- **A process by which we take a chunk and split it up into parts. Each part is called a 'Sub-Chunk'.**
  - **Sub-Chunks were typically split up into the following:**
    - Ground
    - Buildings
    - Props
    - Foliage *(Trees and Bushes)*
    - Interiors
    - Effects
  - **Provided the flexibility needed for a large team of artists to work on different aspects of a chunk in parallel.**
  - **Exporting time went an hour down to minutes.**

# Sub-Chunk Pipeline



## Always Loaded

- Term we use to describe the permanently loaded parts of the city.
- Goal was to push the fog plane out and let the player see for virtual miles to make them feel as if they are part of a massive, dense, urban city.
- Essentially is a low-resolution representation of the high-resolution city.
  - Split up into chunks the exact same as their high-resolution counterparts.

## Always Loaded

- **Consisted of only buildings, major landmarks and ground.**
- **Rendered with very cheap shader:**
  - **Saved on memory.**
  - **The GPU load was light.**
- **When a high-resolution chunk unloaded, the low-resolution representation is flagged internally to render.**
  - **This caused massive popping problems because:**
    - Building silhouettes did not match, the Always Loaded (AL) versions were mostly boxes.
    - Texture resolutions and shader features were different.

# Always Loaded

- **Other problems we encountered were:**
  - **Because of the way stream triggers and chunk borders are designed, the player could sometimes get too close to an Always Loaded chunk and see the low-resolution geometry. To counter this issue we developed two separate solutions:**
    - **Predictive Streaming** – A process by which we would look at the speed and direction of the player. Based on that information, the game would make a *prediction* as to which chunk they are most likely to go into next and start the stream load a few seconds sooner.
    - **Object Flags** – We allowed artists to flag a object with the rule that if "these Fully Loaded chunks and Always Loaded chunks are loaded, keep rendering this object".

# Memory

- Every piece of content in the game has at least two separate files.
  - One that contains mesh data, the other texture data. We do this mainly for streaming purposes. We stream in mesh data first to ensure that there is collision data in case the streaming system falls behind. We will then stream in the texture data afterwards.

## Chunk Memory

- **Each chunk consists of four target files.**
  - **Chunk** – This holds all of the mesh data for the chunk.
  - **Peg –** Contains all of the texture data for the chunk.
  - **Hmap** – The heightmap information for AI helicopters.
  - **PVS** – The pre-computed visibility information for the chunk.

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

# Chunk Memory

- **Xbox 360 memory padding restrictions were problematic.**
  - **The rule is, any texture who has mipmaps and its size is less than 128x128 will automatically pad out in memory to fit that space.**
    - Was unexpected, we did not catch it in the hardware specs early in production.
    - It immediately caused all of our Pegs (textures) to balloon out of control (typically by a factor of three) and not fit into memory.
    - Implemented a method called ‘MipMap Interleaving’.
      - A process in which we analyze the wasted space used in the Peg file and rearrange them in memory to fill in that space.
      - In order for this to work, a texture must abide by certain rules.

# Other Memory Issues

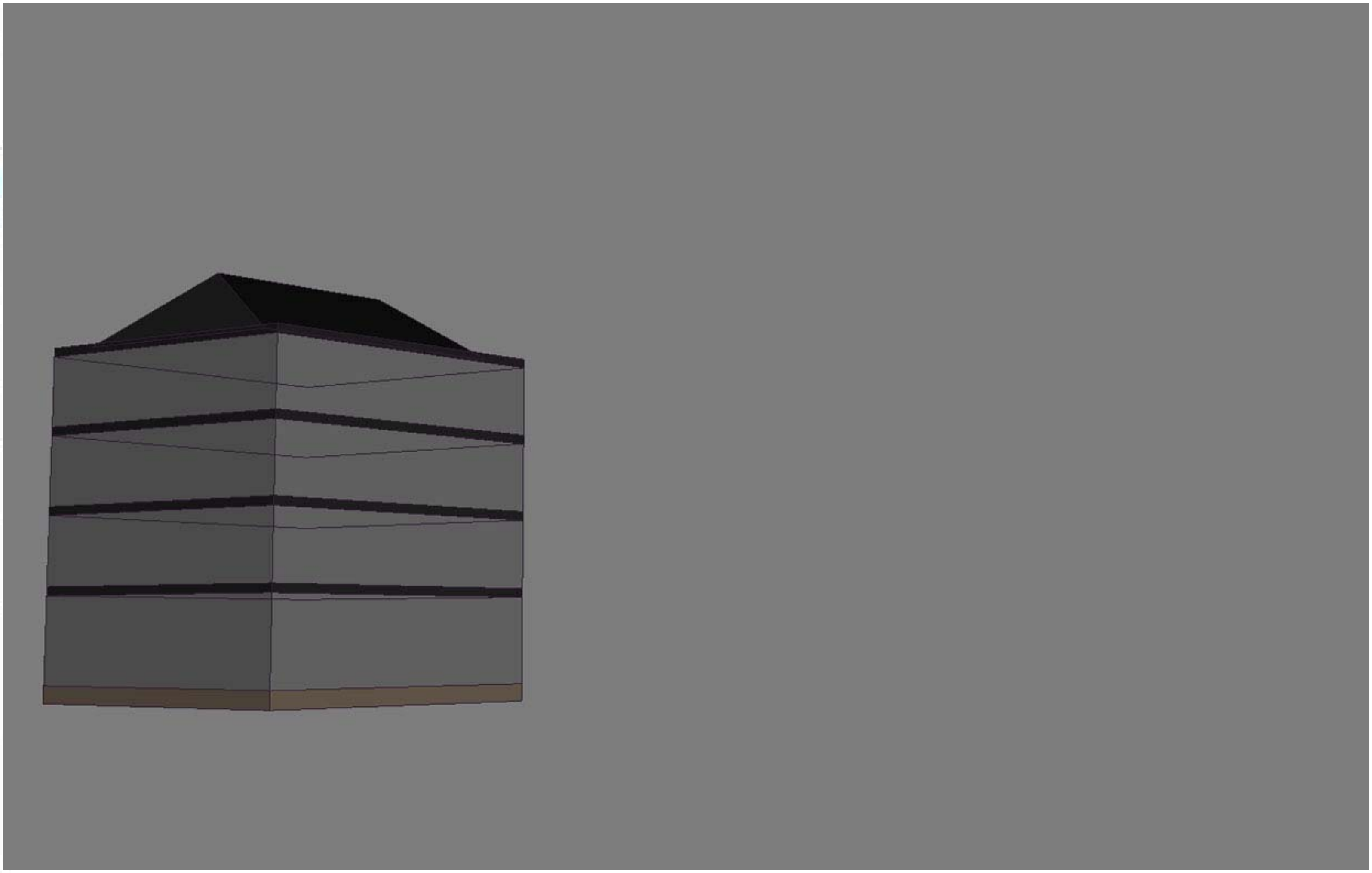
- **The typical shader used three different types of textures.**
  - Diffuse Map
  - Normal Map
  - Specular Map
- **If the shader supported more than one UV channel, we paid the memory cost of each UV channel.**
- **To save on memory, commonly used textures would be put into a common mempool called the AlwaysLoaded_User peg.**

# Buildings

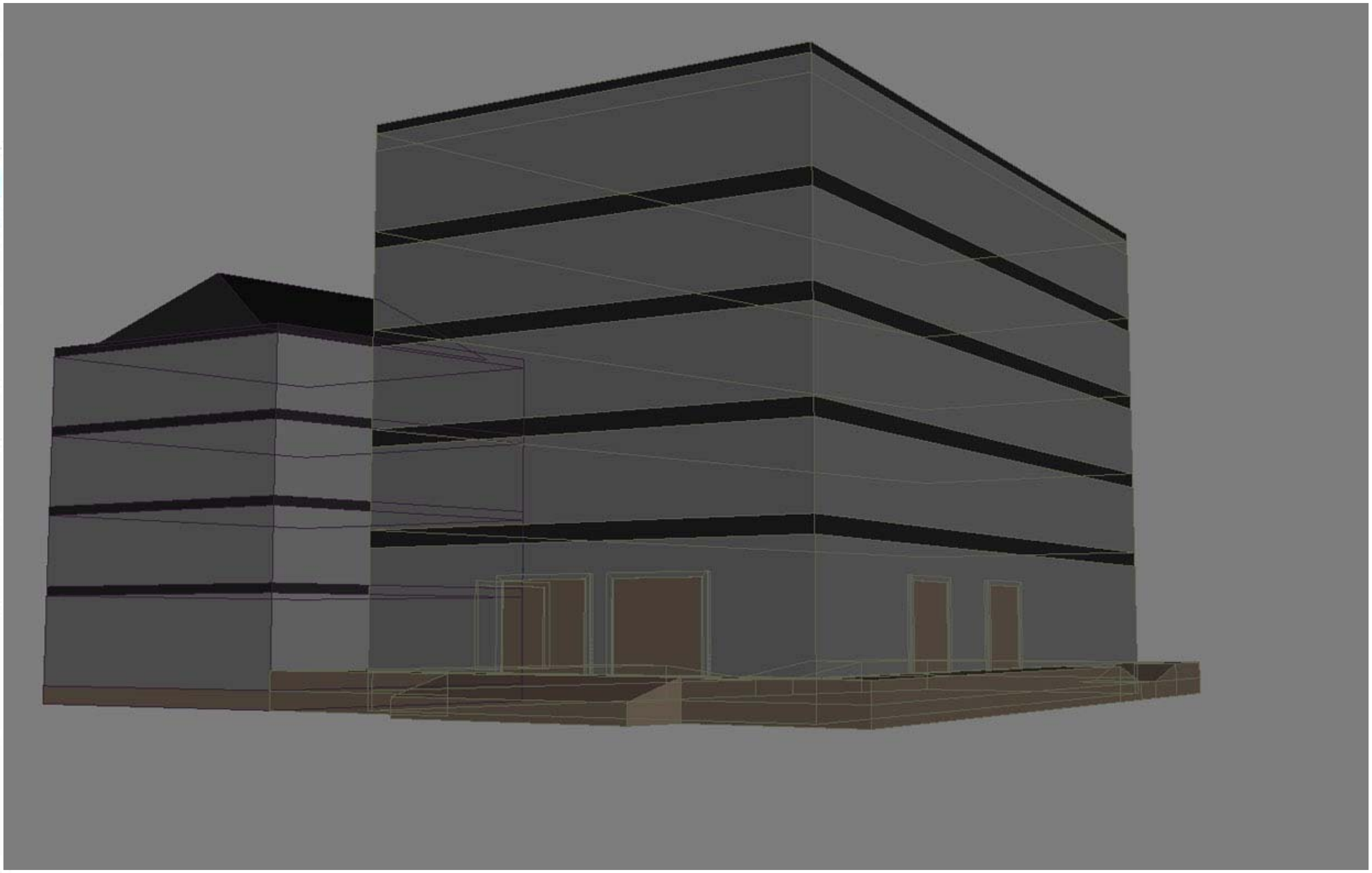
# Buildings

- **Buildings To Be Outsourced**
  - **Four Major Categories**
    - Landmark Buildings
    - Enterable Shops
    - Enterable Strongholds
    - Generic Filler buildings
  - **Categories Sorted by Complexity**
    - Complexity Was Measured in Work Units
      - One Unit Represented One Weeks Work
    - Prior To Outsourcing Each Building Was Rough Modeled

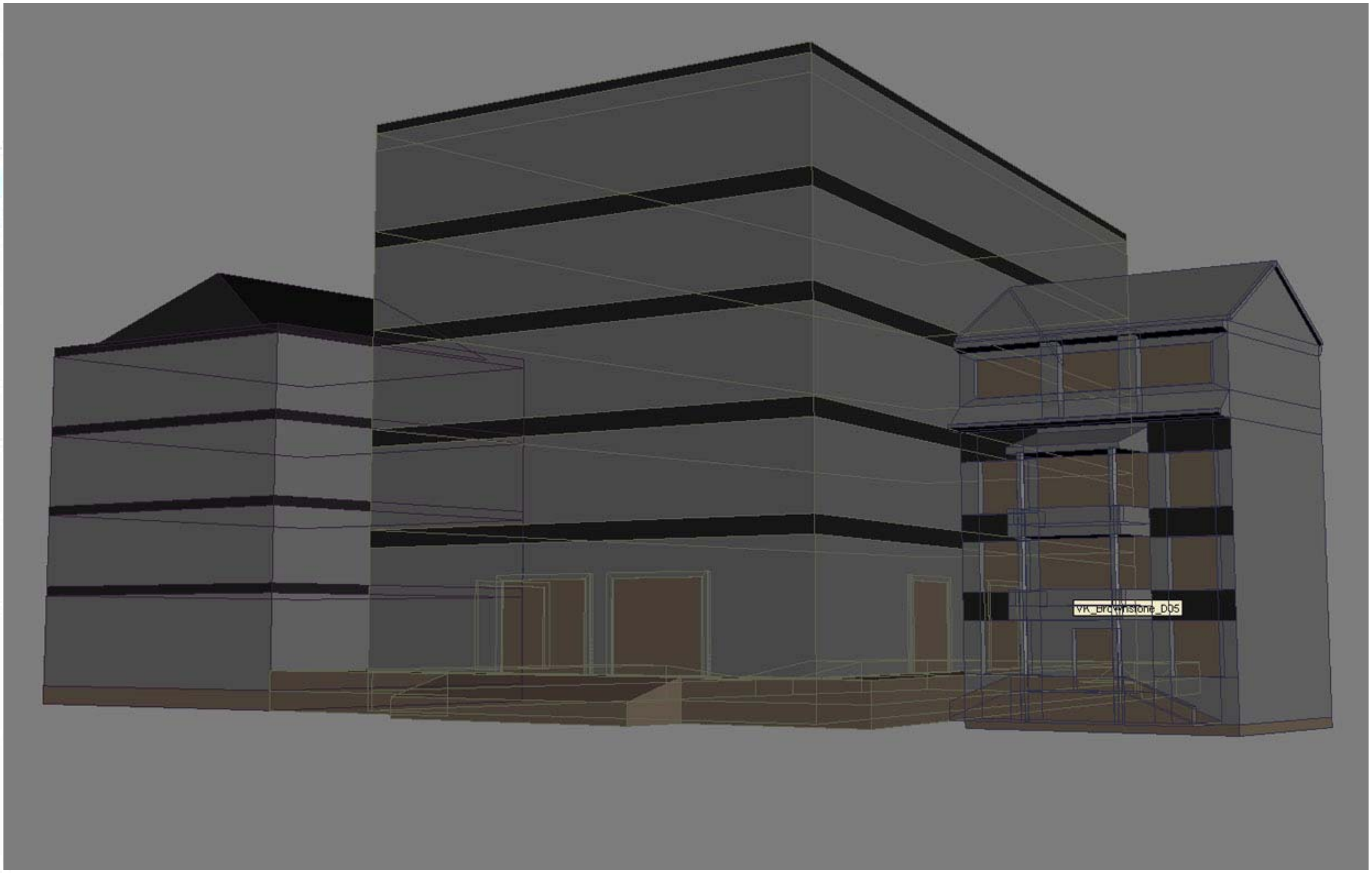
# Three Proxy Detail Levels



# Three Proxy Detail Levels



# Three Proxy Detail Levels



# More About Buildings

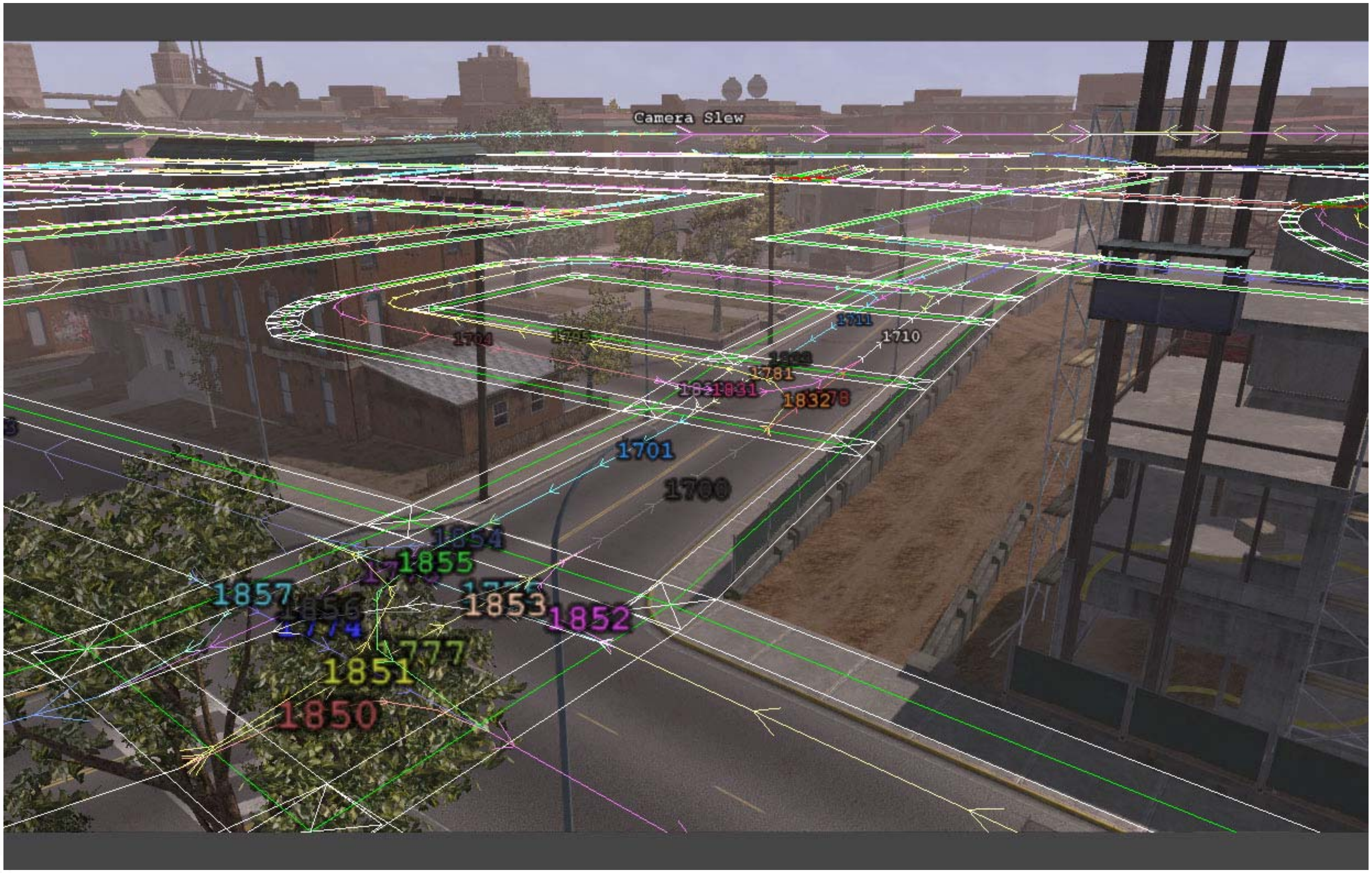
- More Detailed Buildings Worked Better
  - Conveyed Final Look More Easily
  - Allowed For More Accurate Placement
  - Ensured Doors of Enterable Buildings Worked w/ Gameplay
- Final Building Schedule
  - +-300 Units of Work
  - 6 Man Years
  - Over One Hundred Buildings
  - Ten Months of Production Left and No Time to Make Them
  - Outsourcing All Buildings Necessary

# Example of Shader Features

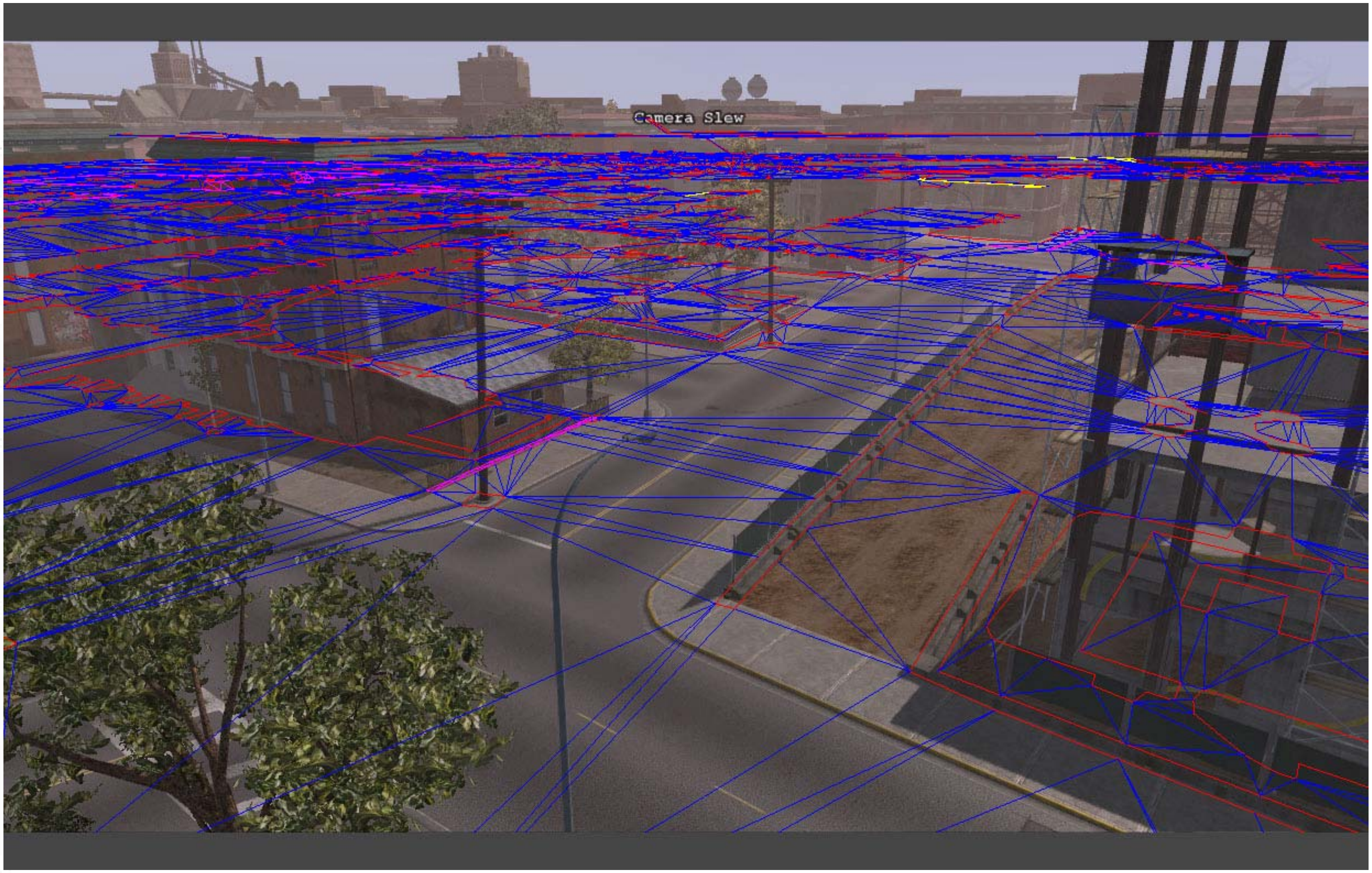
# Stage Two

- Stage Two Construction Goals
  - Functioning Well Enough to Prototype Gameplay
  - Visualized With Neighborhood Themes Conveyed

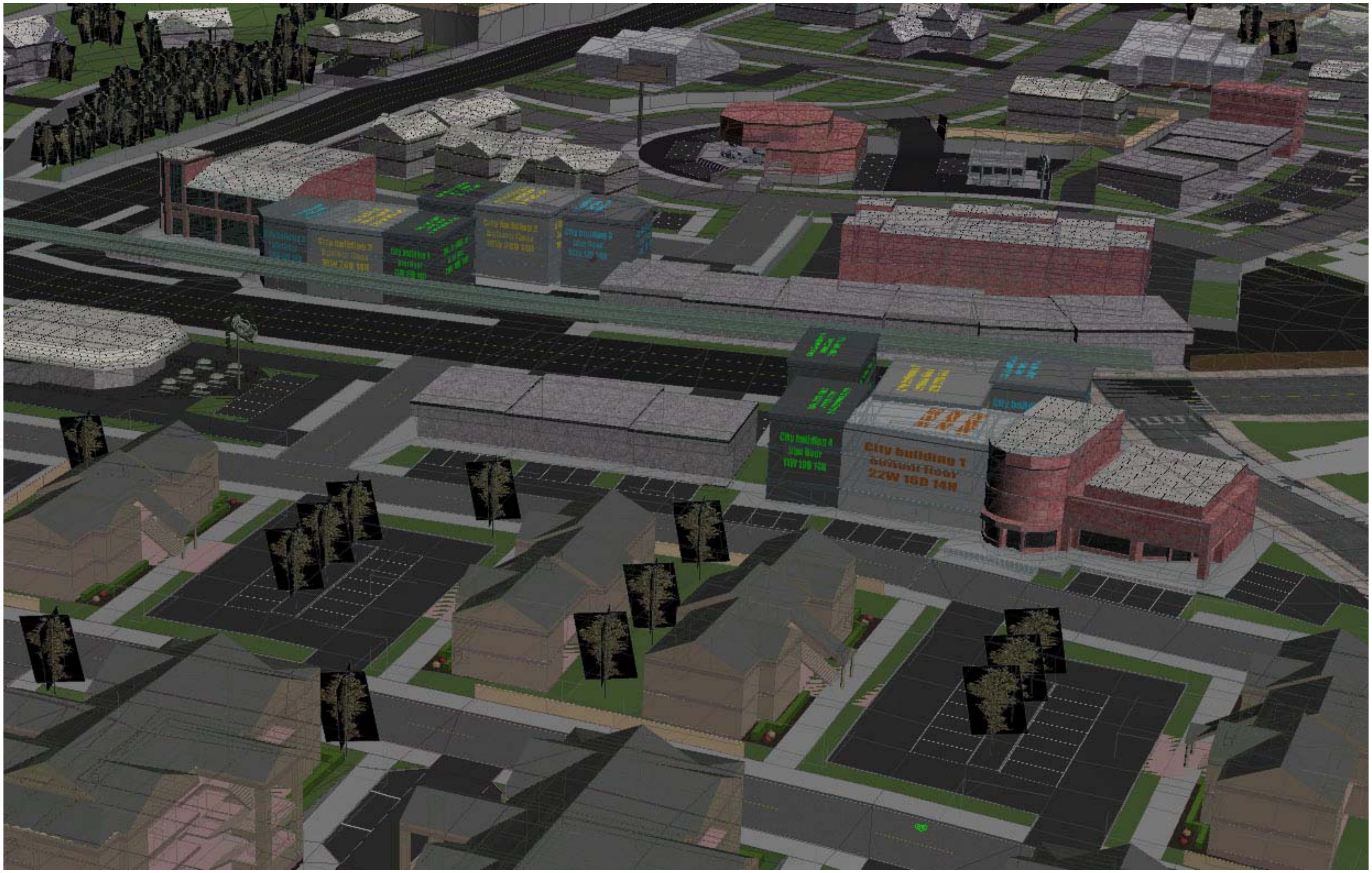
# Navigation Splines



# World Navigation Splines



# Stage Two Results



# Red Light Neighborhood To Be Demoed

# E3

- **E3 Demo Issues**
  - **Streaming Was Difficult to Work With**
  - **Redlight Was Taking Longer to Complete Than Expected**
    - Added More Team Member to Demo Work
  - **Level Was Not Fitting Into Memory**
    - 150% Over Budget

# E3 Solutions

- We Managed to Fit the E3 Demo into Memory
  - Microsoft Doubled the 360's Ram Just Before E3
    - Still Some 60mb Over
  - Scrapped the Streaming  Demo Plan in Favor of One Huge Memory Pool
  - Cut the Total Number of Unique Buildings in Half
    - Down To 13
  - We Didn't Do Any of the Building Variation Planned
  - Stole From All Other Available Memory Pools

# Saints Row vs. GTA: SA


# Post E3

- The E3 Demo Was Complete
  - Desperately Needed a Realistic Schedule
  - Memory for the Demo Was Way Over Budget
  - Streaming Still Had Not Gotten a Valid Test
  - Frame Rate Was not Shippable

# Realistic Schedule Creation

- All Environment Artists Meet to Evaluate the World
  - Each Block Was Reviewed
  - 400 Blocks
  - 3 Days
- Block Estimates Where Dropped into a Schedule
  - Current Date Was June ’05
  - Content Complete Was Scheduled for January
  - The City Schedule Was Nearly 20 Man Years of Work
  - The City Schedule Stretched Out to March ‘07

# A Plan to Finish The City

- **Scale Back the Amount of Unique Art**
- **Cut City Size Where Possible**
- **Outsource More**
  - **Interiors**
  - **Assign the Majority of Internal Artist to Exterior Ground Creation**
    - 16 of 20 Primarily Tasked with Exterior Ground Related Work
- **Flatten the World**

# The Last Push

- Our Last Hurdles to Finish The World
  - Fitting the Chunks into Memory
  - Making Streaming Work
  - Getting Frame Rate Up
- Memory Issues
  - To Much Unique Art
    - Reduced the Number of Unique Buildings Per Chunk
      - Originally Had 20-30 Unique Buildings Per Chunk
      - Reduced to 12-13
    - Down Resed Textures, Consolidated Others, Etc.
  - When All Else Failed the Chunk Would Be Split in Half
    - Created Streaming Problems

# The Last Push

- **Streaming Issues**
  - **The Player Could Get to a Chunk Before it was Loaded**
    - We Would Try and Place Obstacles to Slow the Player Down
    - As a Last Resort a Road Would Be Blocked Off
      - This Would Cause Lots of Rework
- **The Last Challenge**
  - **Frame Rate**

# Performance

- Shaders
- Shader LOD
- Draw Calls
- Occlusion
- PVS (Pre-Computed Visibility)

# Shaders

- Using hardware shaders in the art pipeline was a first for the studio.
- Technical Artists created the shaders.  Rendering programmers would do any optimizations to the shader necessary.
- Because we didn't have any beta/final hardware for a long time, we were developing the game on alpha hardware and the PC.  It was unclear how much of an impact shader ALU and features would impact framerate.

# Shaders

- **Shaders**
  - **Fill rate was not a problem for us during the day time. At night when headlights and street lights came on is when it became a problem. This is when shader features became really important.**
  - **Framerate was subjective to the current view and the limiting factors were typically:**
    - Features in the shader (i.e. expensive ALU, number of texture lookups)
    - The current frame's draw call count
    - Varying number of shaders
      - There is a CPU cost associated with uploading a new shader to the GPU.

# Shader LOD

- **A system by which we could, at certain distances, effectively ‘turn off’ features of a shader .**
  - **Helped out on fillrate during night time.**
  - **Accomplished by:**
    - Each shader had its own shader lod rule file. In the rule file, for each LOD, we could dictate what features of the shader should be active.
    - Since there is no way to remove features of a shader at run-time, our shader cruncher would create LOD versions of the shader based on the rules we set up.
    - At a distance of every 30 meters, the game would swap the current objects shader with the appropriate LOD version.

# Shader LOD

- **Example of what we would remove at each LOD on a typical shader.**

| 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 Calls

- **Because of the object density, we often ran into command buffer overflows.**
  - **The typical side-effect was objects dropping in and out.**
- **An average object count on any given frame could easily be in the thousands.**
  - **This forced us to develop a PVS (Pre-Computed Visibility) system late in production.**
  - **With PVS, the average object count was around 800 – 1000 on a typical frame.**

# Occlusion

- **Before PVS, we primarily used occluders as the primary way of occlusion.**
  - **Our occluders consist of primitives shaped as either boxes or an isoceles triangle.**
    - They were mostly put inside of buildings, walls and other large objects.
    - It was clear in production that this type of occluder system did not lend itself well to an open world environment.
    - This type of occlusion works well for linear level designs.
      - Broke down especially when the player looked down long streets, or standing on the shore line looking across the river where hundreds of objects are in view with nothing to occlude them.

# Occlusion

- **We ended up developing a hybrid occlusion system.**
  - **Occluders would be left on as a fail safe if PVS dropped out.**
  - **They would also occlude dynamic objects such as:**
    - Vehicles
    - Pedestrians
    - Foliage (trees and bushes)
    - Props (dynamic lampposts, benches, etc.)

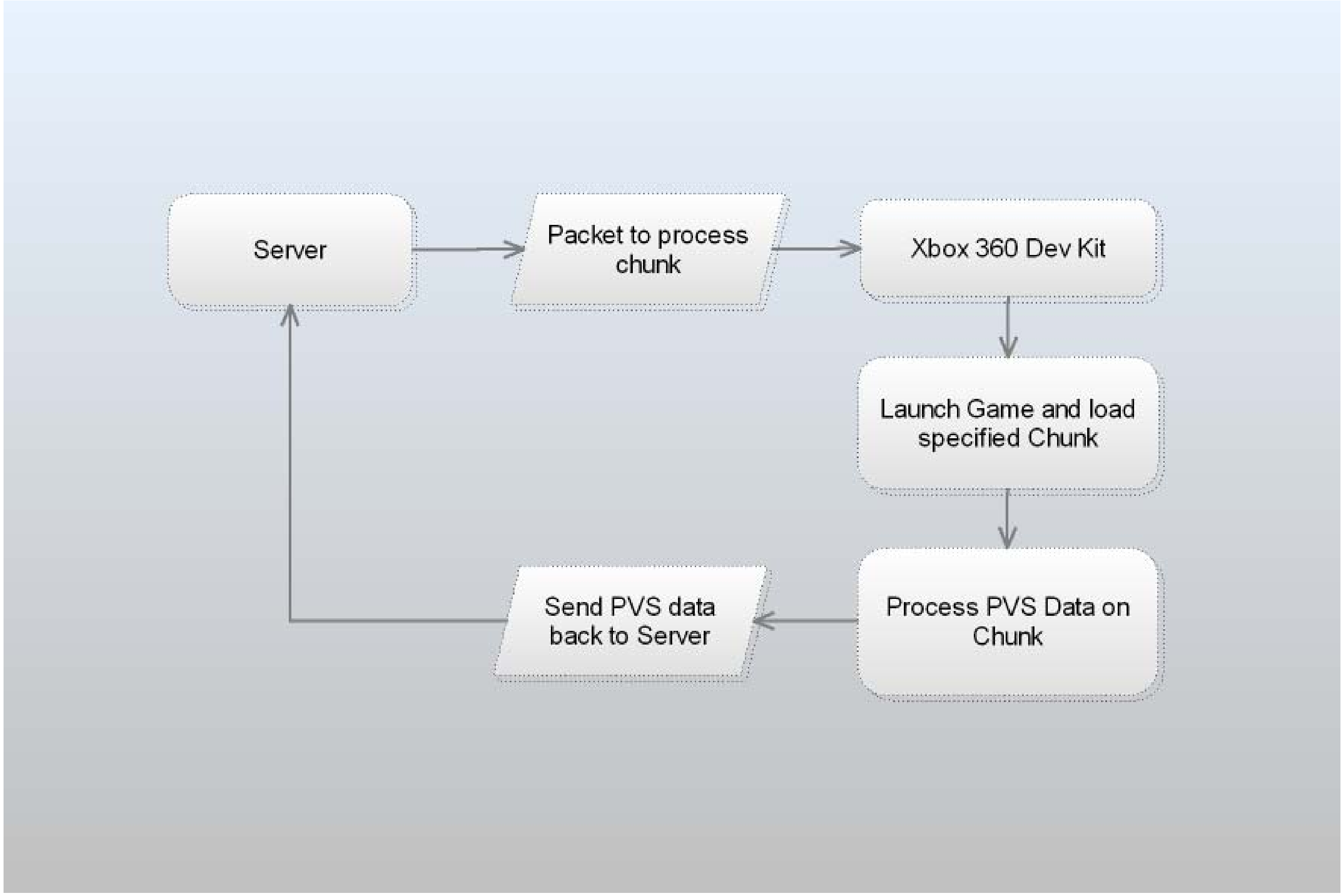
# Pre-Computed Visibility (PVS)

- This system would only occlude static level mesh geometry.
- Developed very rapidly about 2 months out before we submitted to Microsoft.
  - One month of development.
  - One month of bug fixing.
- Was our last ditch effort to get the game running at 30 FPS in normal gameplay.

# Pre-Computed Visibility (PVS)

- **Calculating PVS was an automated process across a farm of Xbox 360 Dev Kits. The steps involved were:**
  - Wrote an app that would send packets to various Dev kits with information about which chunk to process.
  - When the Dev kit received the packet, the game would launch and start processing PVS data.
    - A chunk would be mathematically subdivided into grid cells.
    - Within each grid cell, the camera would sample a number of pre-determined points.
    - At each point, the PVS system would sample visible objects in 360-degrees at the pixel level. Any pixel of an object that was visible would be flagged to render.
    - End result was a list of objects to render in that grid cell.
    - Once finished, the PVS system would send the data back to the server to be attached to the chunk.

# Pre-Computed Visibility (PVS)



# Pre-Computed Visibility (PVS)

- **Major issue we ran into with this method of PVS was:**
  - **Would routinely have what we called "object drop out".**
    - Happens when you enter a PVS grid cell and objects that are clearly in view suddenly disappear.
    - To correct this, we had to do "spot fixes".
      - This was accomplished by slewing to that grid cell in the game and generate PVS data on that cell. The information would be saved off to disk and later re-integrated into the chunks PVS data.
- **In the end, PVS helped us achieve a sustainable 30 FPS.**

# Thank you!

# Q & A