Building Blocks
Artist Driven Procedural Buildings

James Golding - Epic Games

Who Am I

- Started as Field Engineer at MathEngine
  - Oxford, UK
  - 1999-2003
- Senior Programmer at Epic Games
  - Raleigh, NC, USA
  - Worked here for nearly 8 years
  - Physics, animation, tools, gameplay…
  - Shipped some games
    - Unreal Tournament 2003, 2004, UT3,
    - Gears of War 1 & 2

Who Is This Talk For

- Programmers
- Level Designers
- Technical Artists

- There is no code, I promise!

- Anyone who thinks about building big cities for games

Our Goals

- Good looking buildings with high visual density
- Easily change shape and size for gameplay
- Automatically generate LODs

Not Our Goals

- We are not interested in generating entire city with one button, or even an entire building, but decorating a building defined by designer.

Existing Approaches

- Use level geometry tools, cover with meshes
  - Lots of work placing meshes
  - Painful to change meshing
- Build custom building meshes
  - Hard to adjust for gameplay
  - Each one needs LOD custom made
- Simple shapes with tiling material
  - Did not meet our visual bar

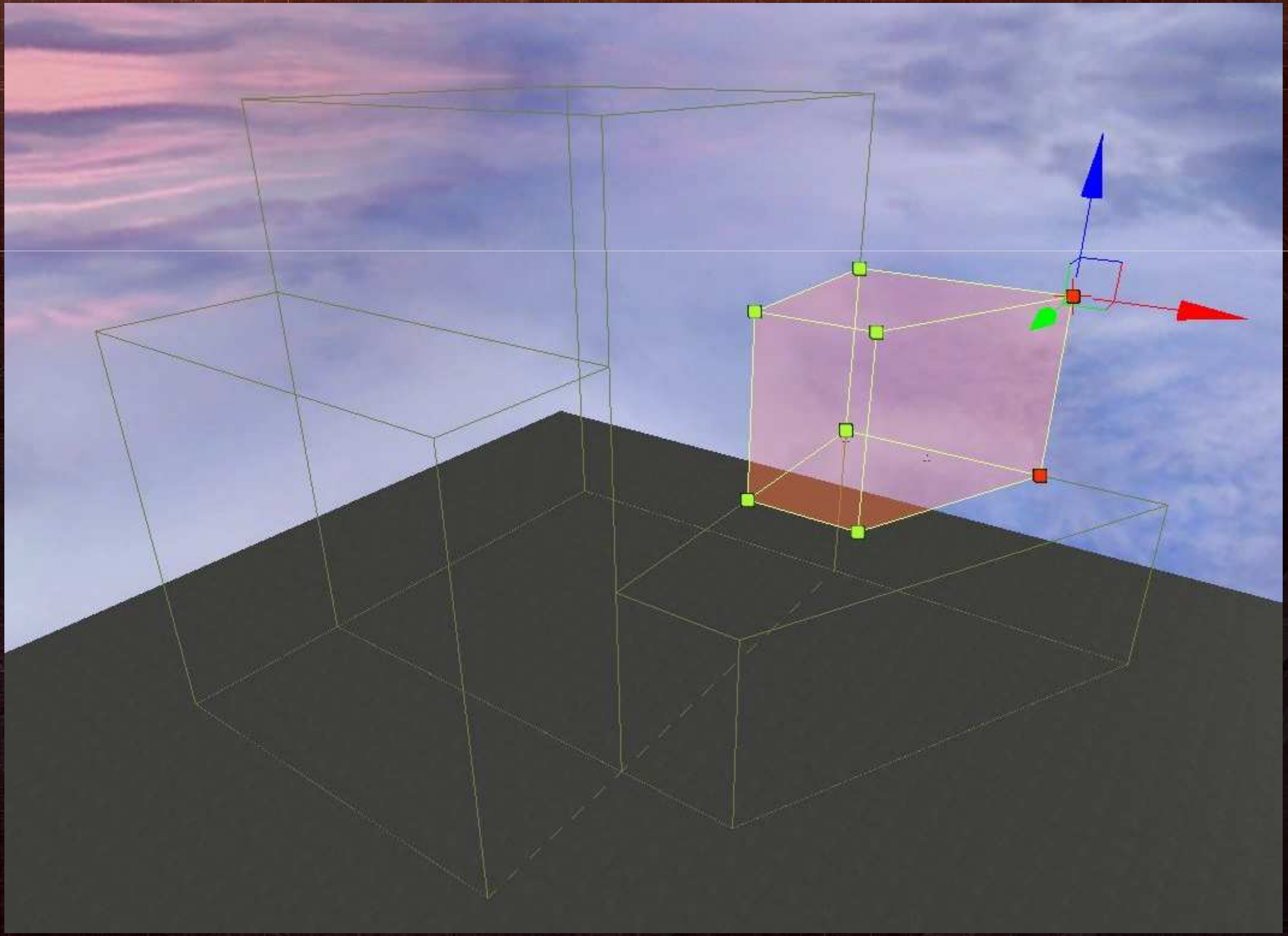
New Approach

- Designer creates 'high level' description of building
- Artists build a library of rectangular, modular, facade meshes
- Artist creates 'ruleset' which describes how facade pieces are used

Initial idea from "Procedural Modeling of Buildings" by Müller et al. (2006, ETH Zurich)

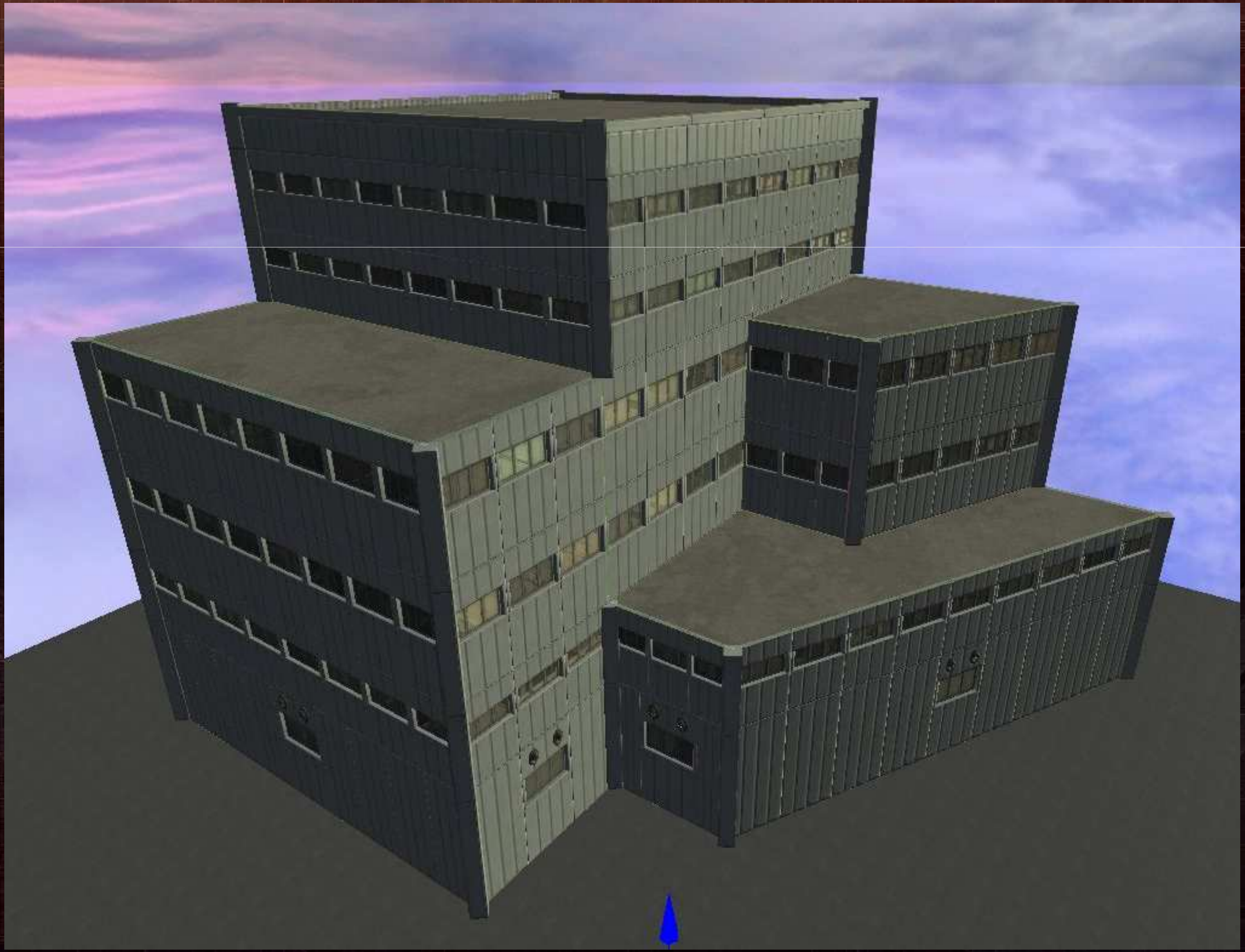
Defining Building Shape

- Collection of simple shapes



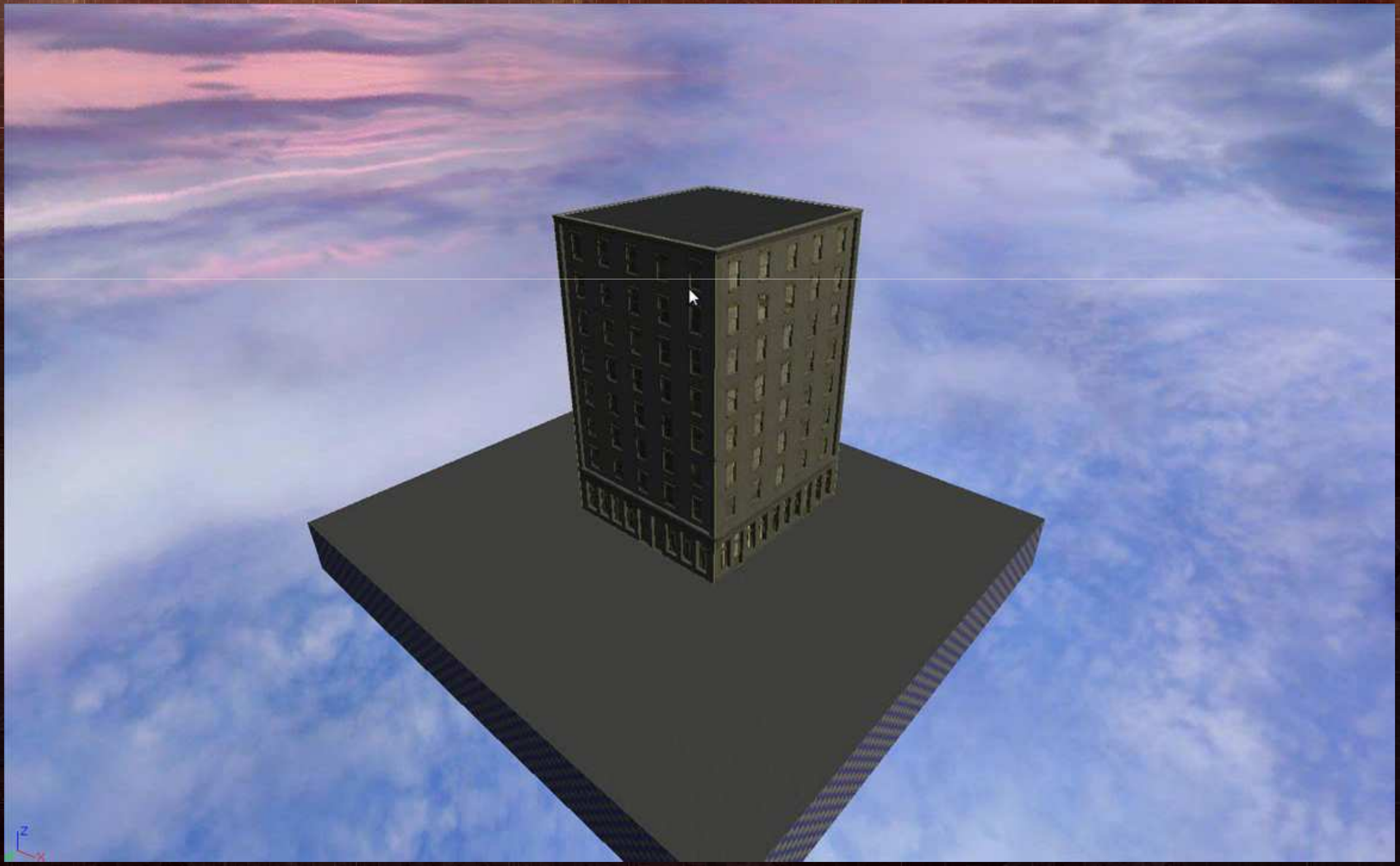
Defining Building Shape

- Apply ‘Ruleset’ to group



Defining Building Shape

- Can easily modify building at any time

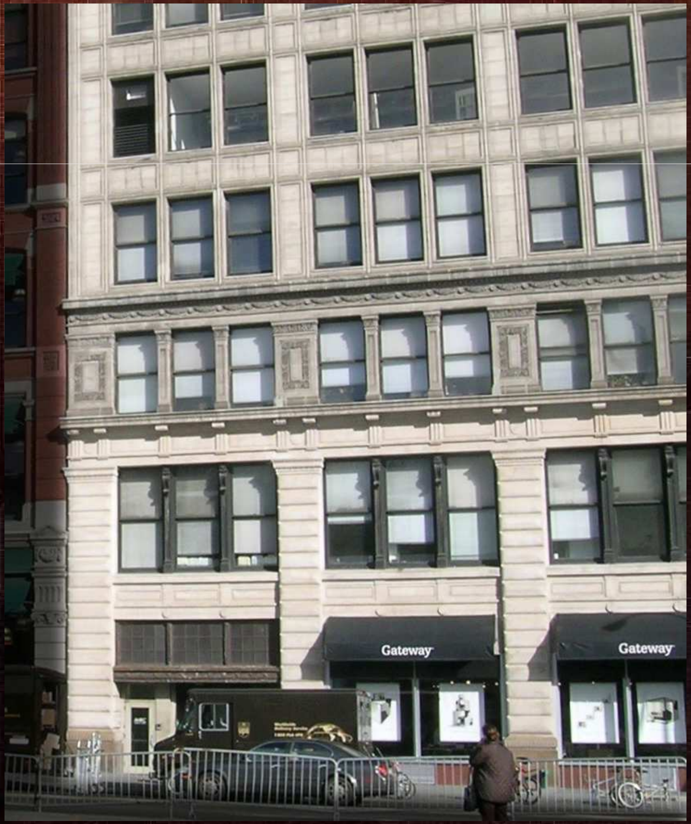


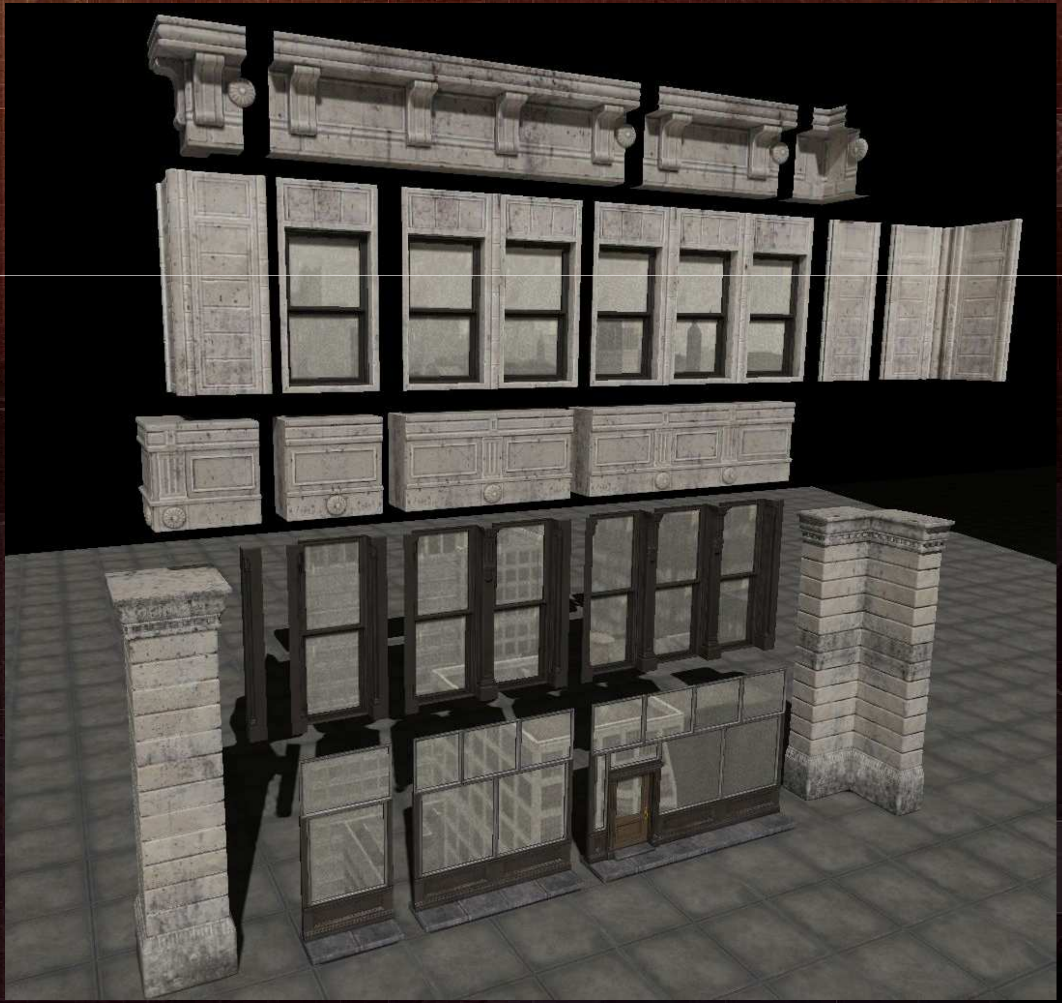
Breaking It Down

- Starting with a reference photo

Breaking It Down

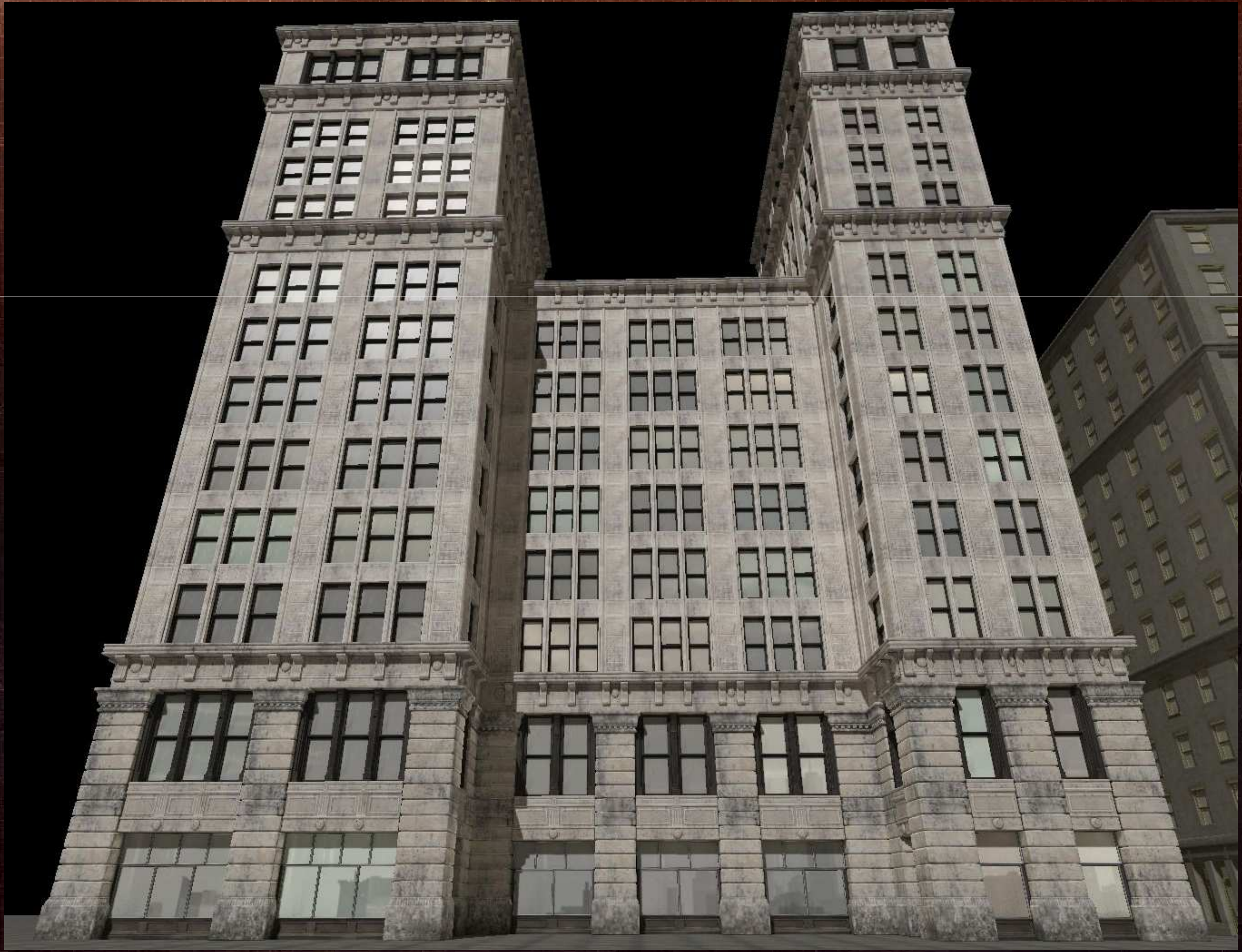
- Artist breaks it into modular meshes





Breaking It Down

- Procedural system places meshes



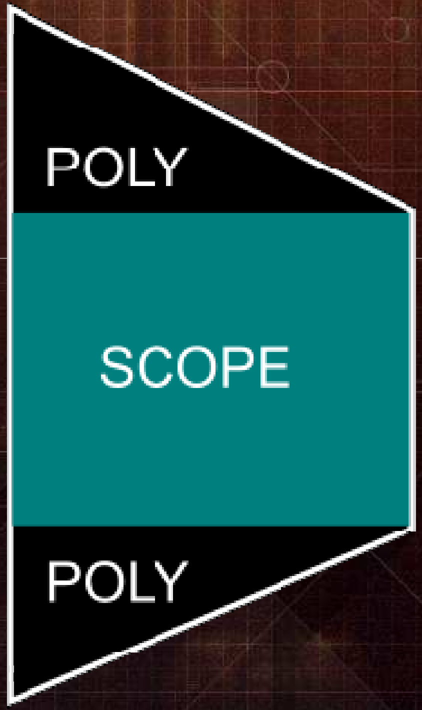
‘Scopes’

- A ‘scope’ is a 2D rectangle
  - Location
  - Orientation
  - Dimensions
- Tool takes 3D building shape and extracts set of scopes

Extracting Rectangles

- Certain areas are not rectangular
  - – Walls - if roof is not flat
  - – Roof - if building plan is non-rectangular
- We make simple polygons to fill holes
- Don’t extract scope from roof
  - – always just big polygon

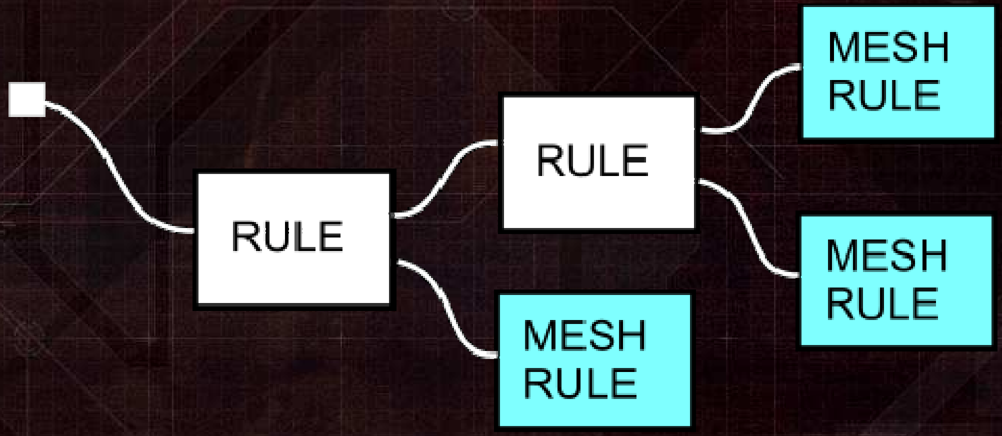
Extracting Rectangles



Rules?

- Each rule can do one of two things:
  - Split a scope into smaller scopes
  - Place mesh that fills the scope area on the building facade

- Forms a graph



Rules?

- A grammar for describing facades
  - ‘Context Free’
- Graph of nodes good for a graphical tool
  - More visual = happy artists

The Rules

Mesh Rule

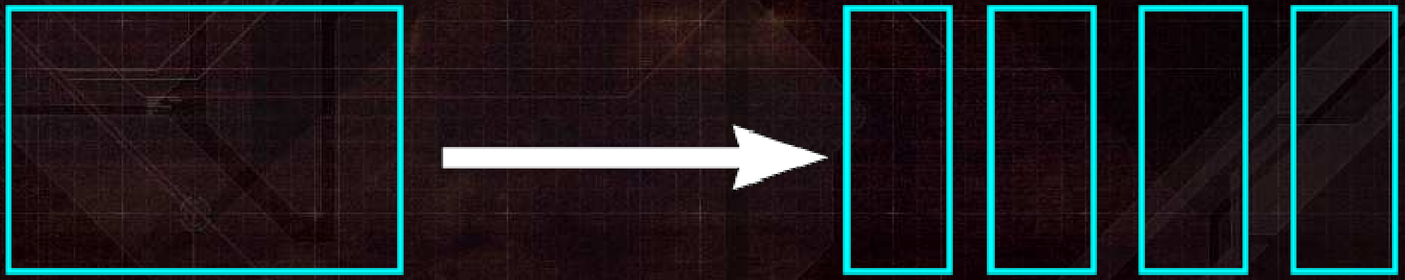
- Artist specifies
  - What mesh
  - X and Z extent that it fills
- Easy to scale and place instance
  - scale = desired size/defined size

Mesh Rule

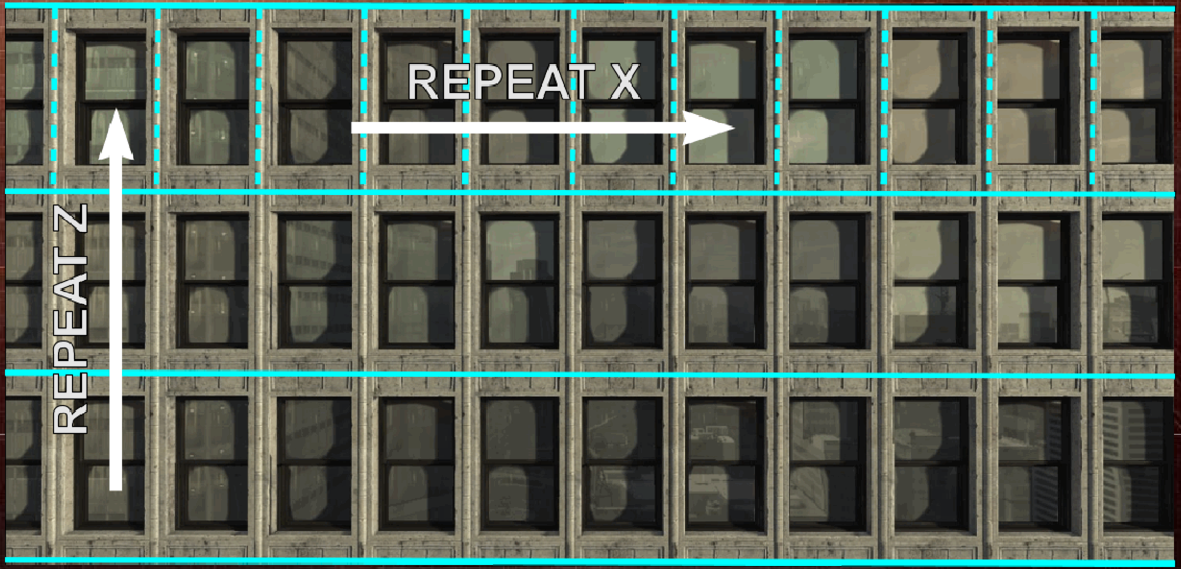
- Initial concerns over scaling of artist built meshes
  - System lets you specify which meshes are scaled and which are not
  - Generally not visually noticeable with building-type meshes
  - Needs to avoid tiny doors etc

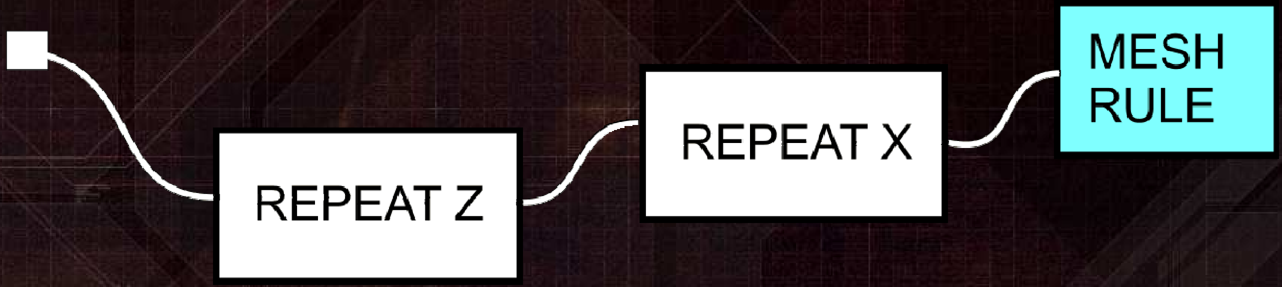
Repeat Rule

- Choose an axis (X or Z)
- Break up scope along that axis into equal size pieces
- Ensure no piece along axis is larger than defined maximum size
- Generates varying number of new scopes, depending on building size



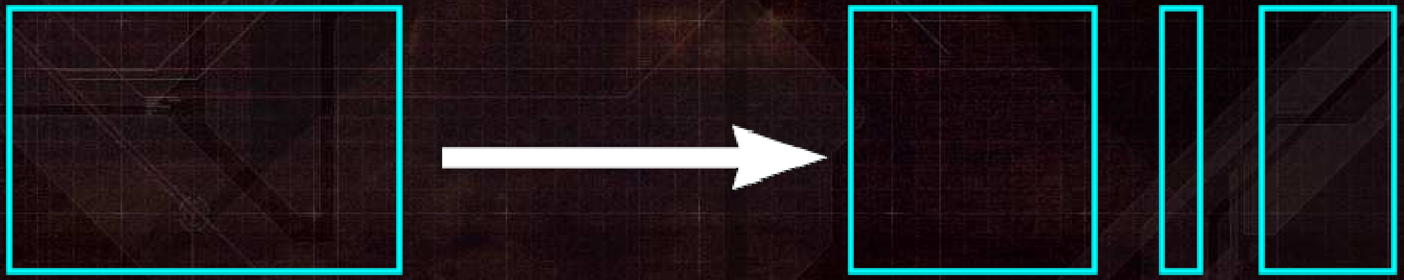
Repeat Rule



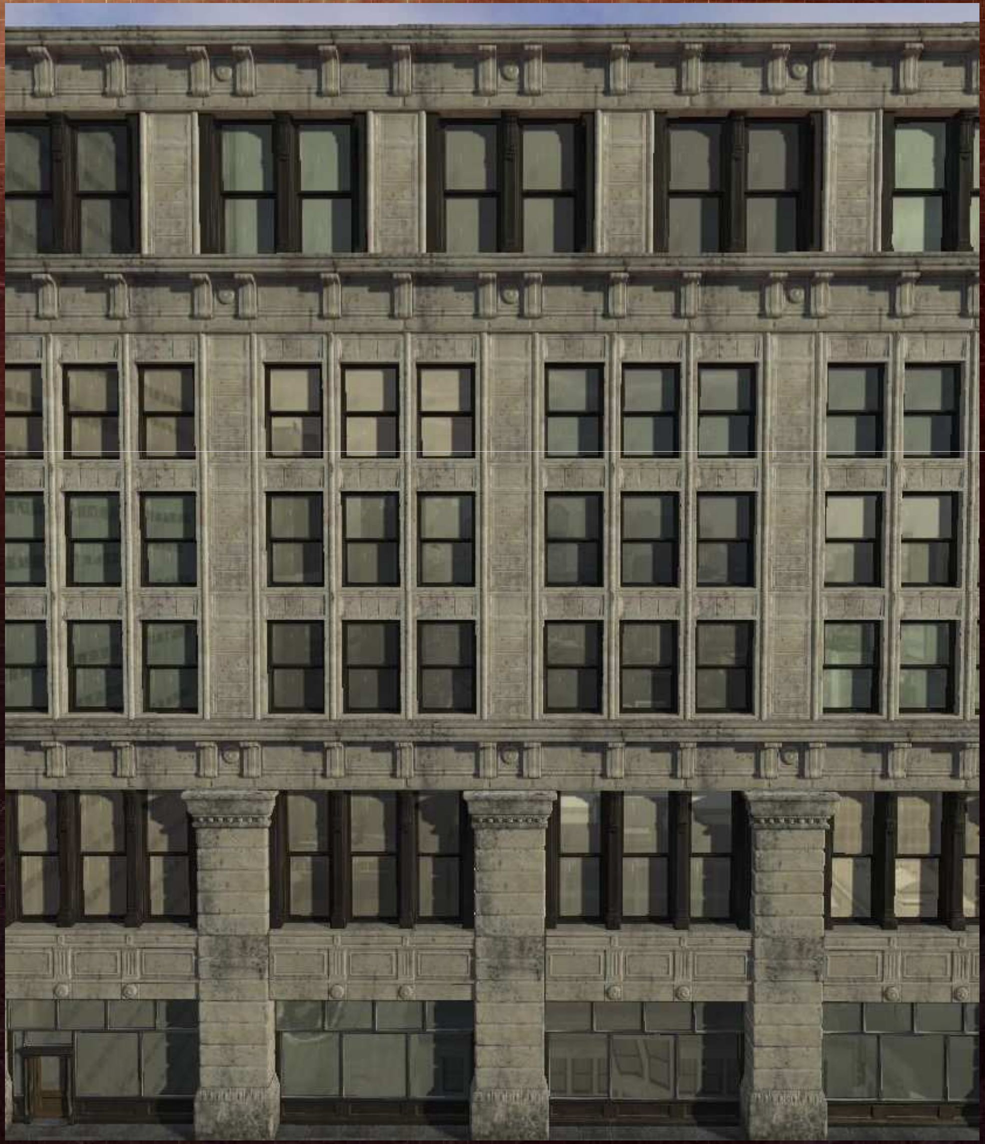


Split Rule

- Designer specifies axis and number of scopes to break into.
- Each split can be fixed size or variable.
- Always require one of the splits to be variable.
- If scope is too small to fit in fixed size areas, must discard them.
- Similar to windowing toolkits (wxSizer etc)

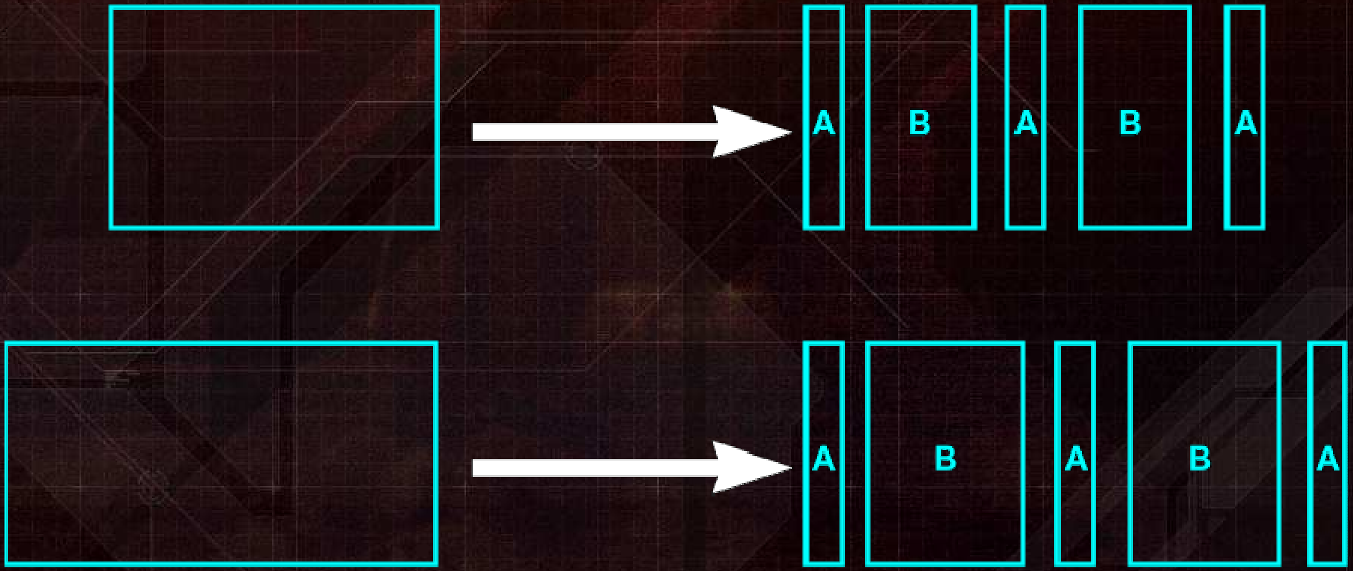


Split Rule

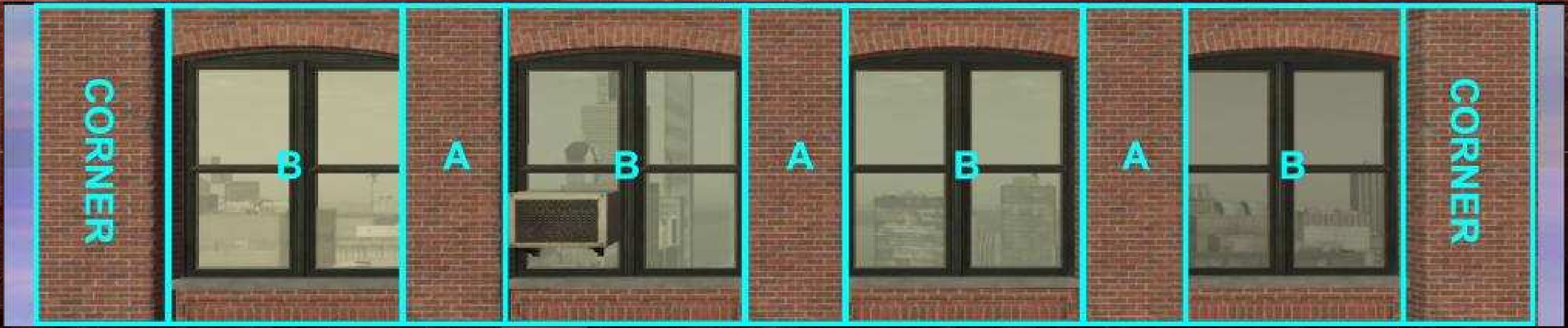


Alternate Rule

- Hard to achieve ABABA 'fence post' layout with just repeat and split.
- A is fixed, B is stretchy.

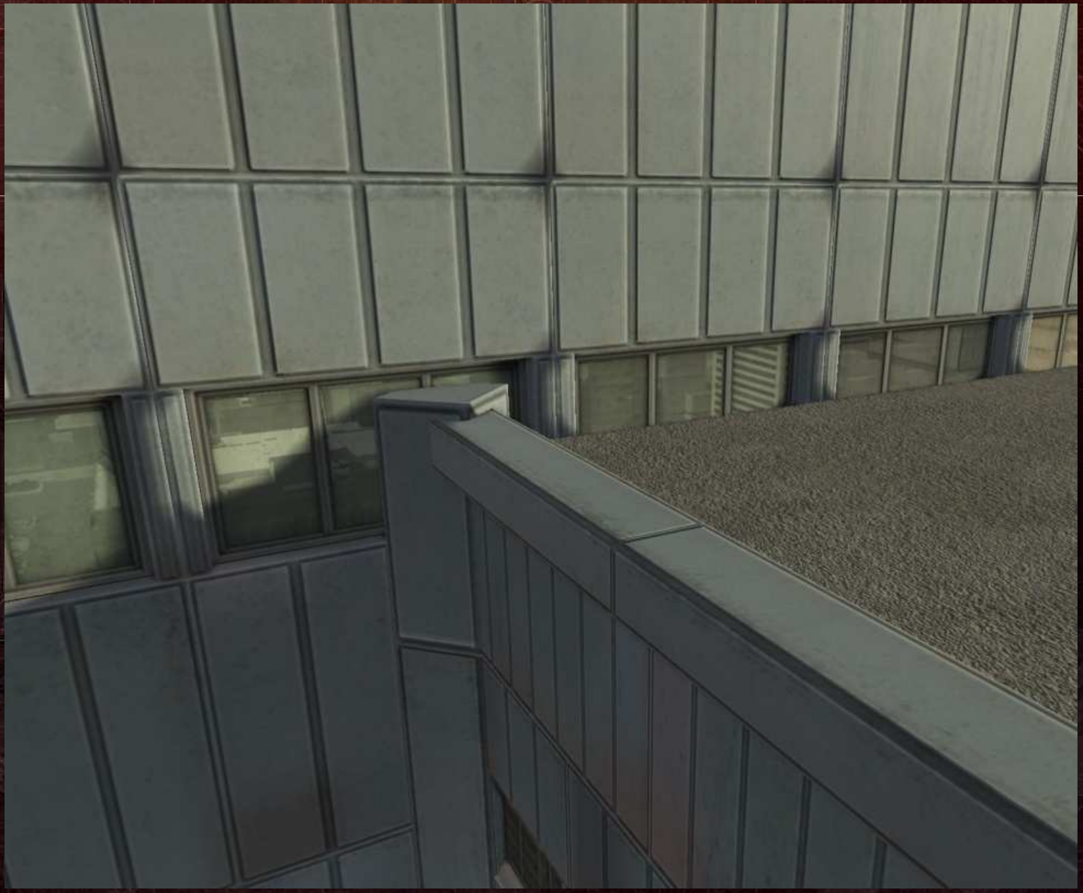


Alternate Rule



Occlusion Rule

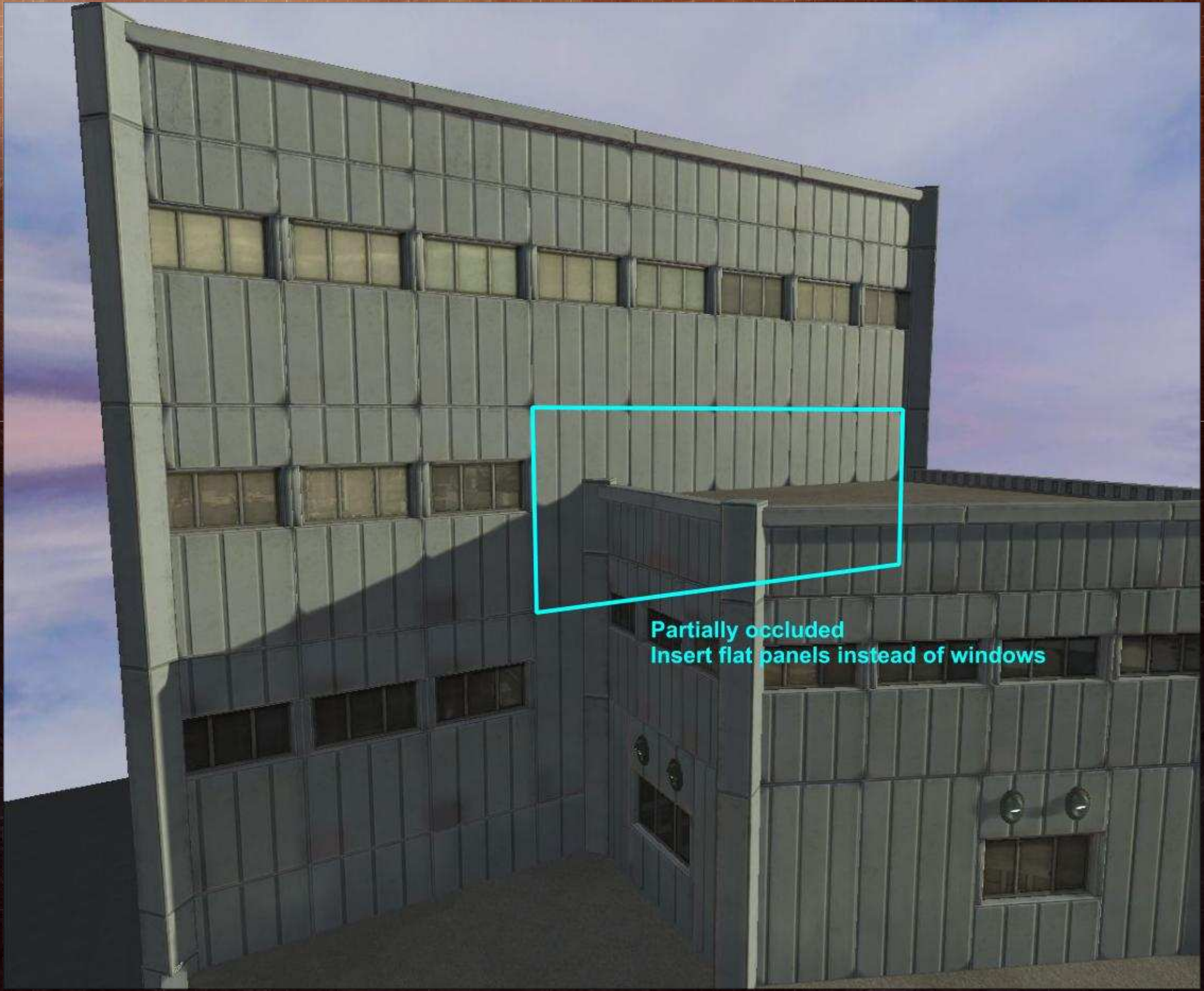
- Quickly find that meshes are being placed where not seen
- Needed for intersections between buildings to look good



Occlusion Rule

- Output is 'clear', 'blocked' or 'partial'
  - Don't place mesh if 'blocked'
  - Can choose different mesh depending on 'clear' and 'partial'
- Initially this was a separate rule node
  - Used so frequently, we included this into the Mesh rule

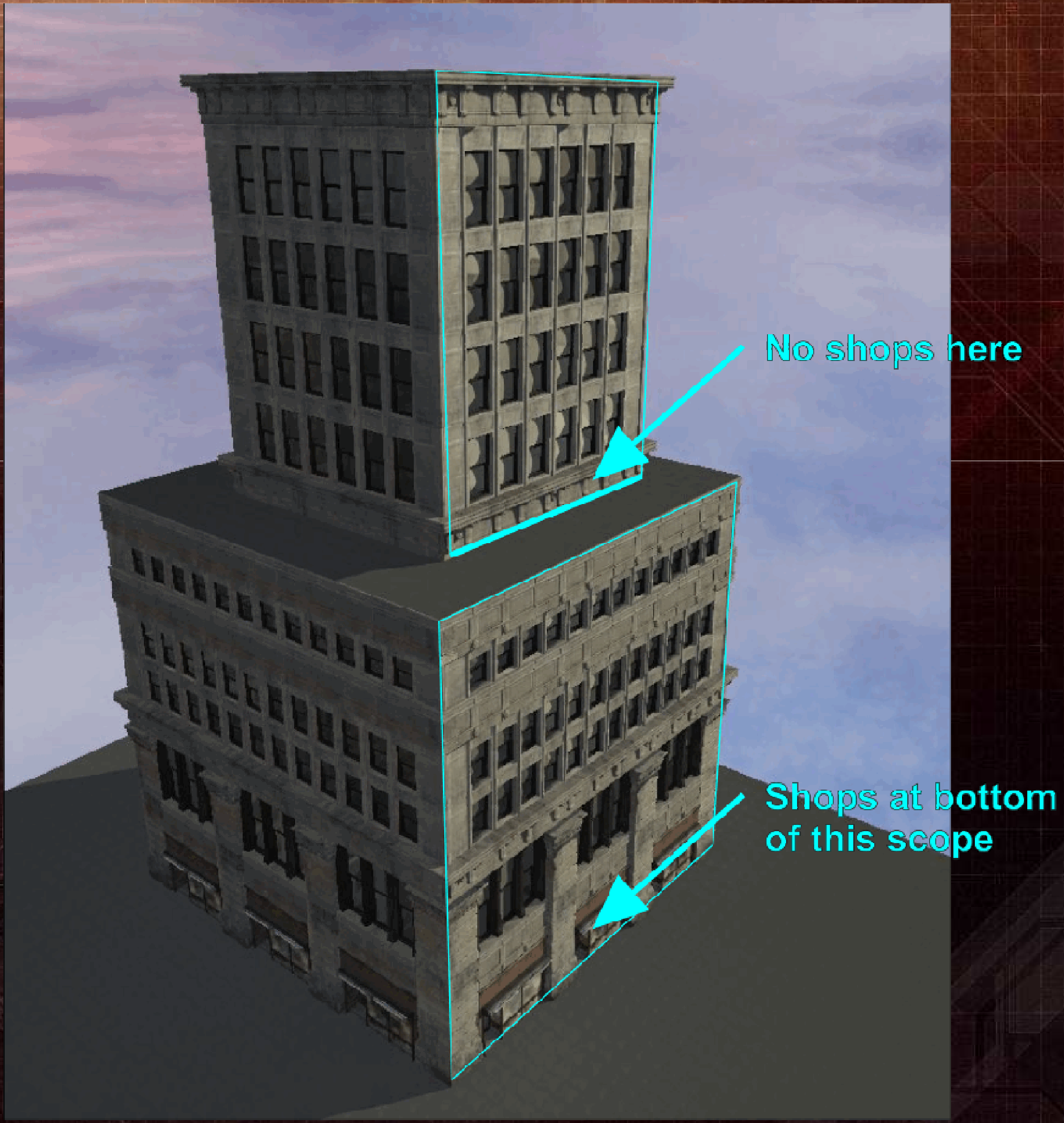
Occlusion Rule



Top/Bottom Rule

- Don't want shop fronts at the bottom of every scope in building
- Performs different actions if bottom of scope is at bottom of entire building
- Does same thing for top (e.g. large trim at very top)

Top/Bottom Rule

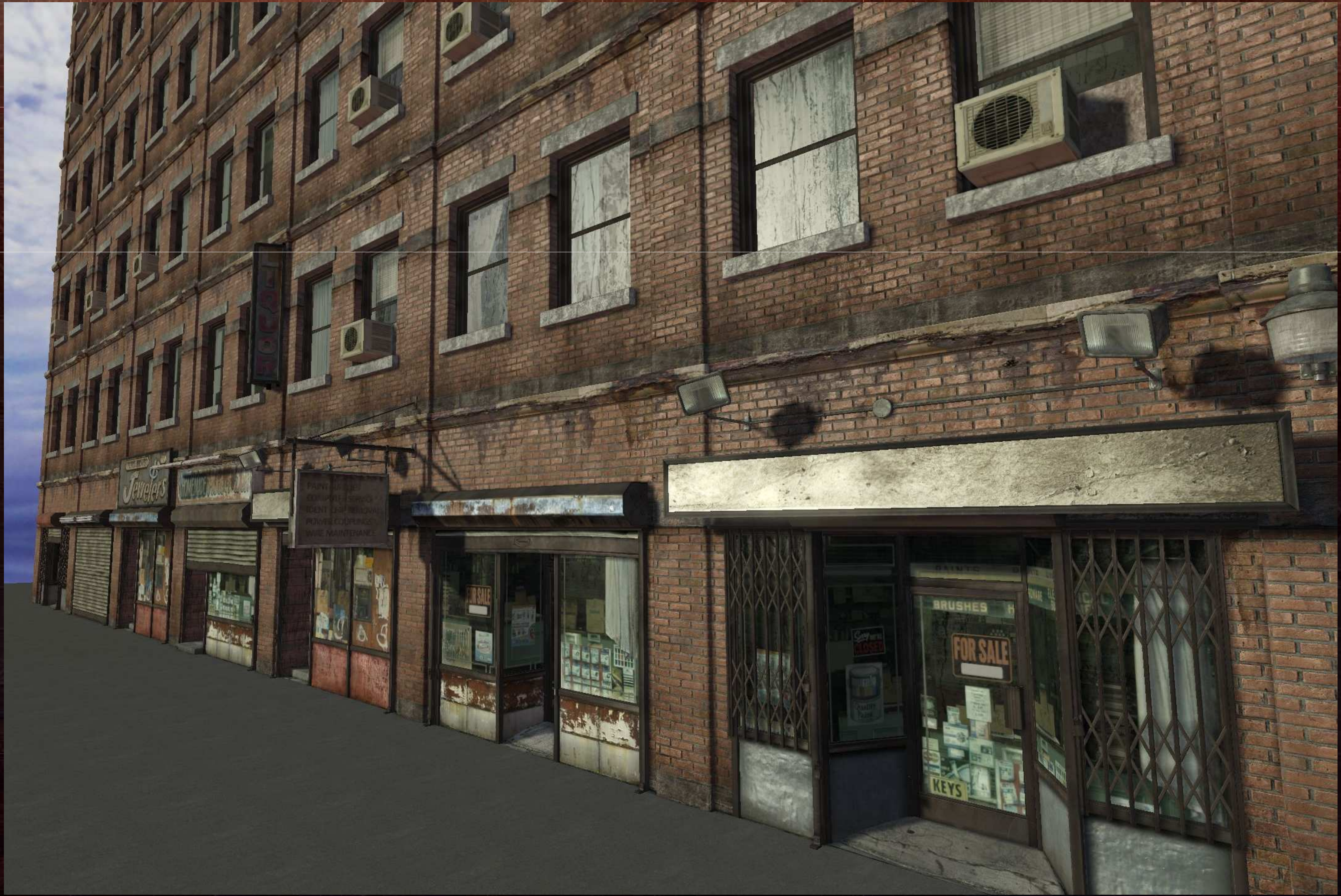


Random Rule

- Does not resize scope
- Executes to N out of the M possible child Rules
- Allows meshes on top of each other
  - E.g. a Window mesh with random AC unit and/or awning.

Random Rule

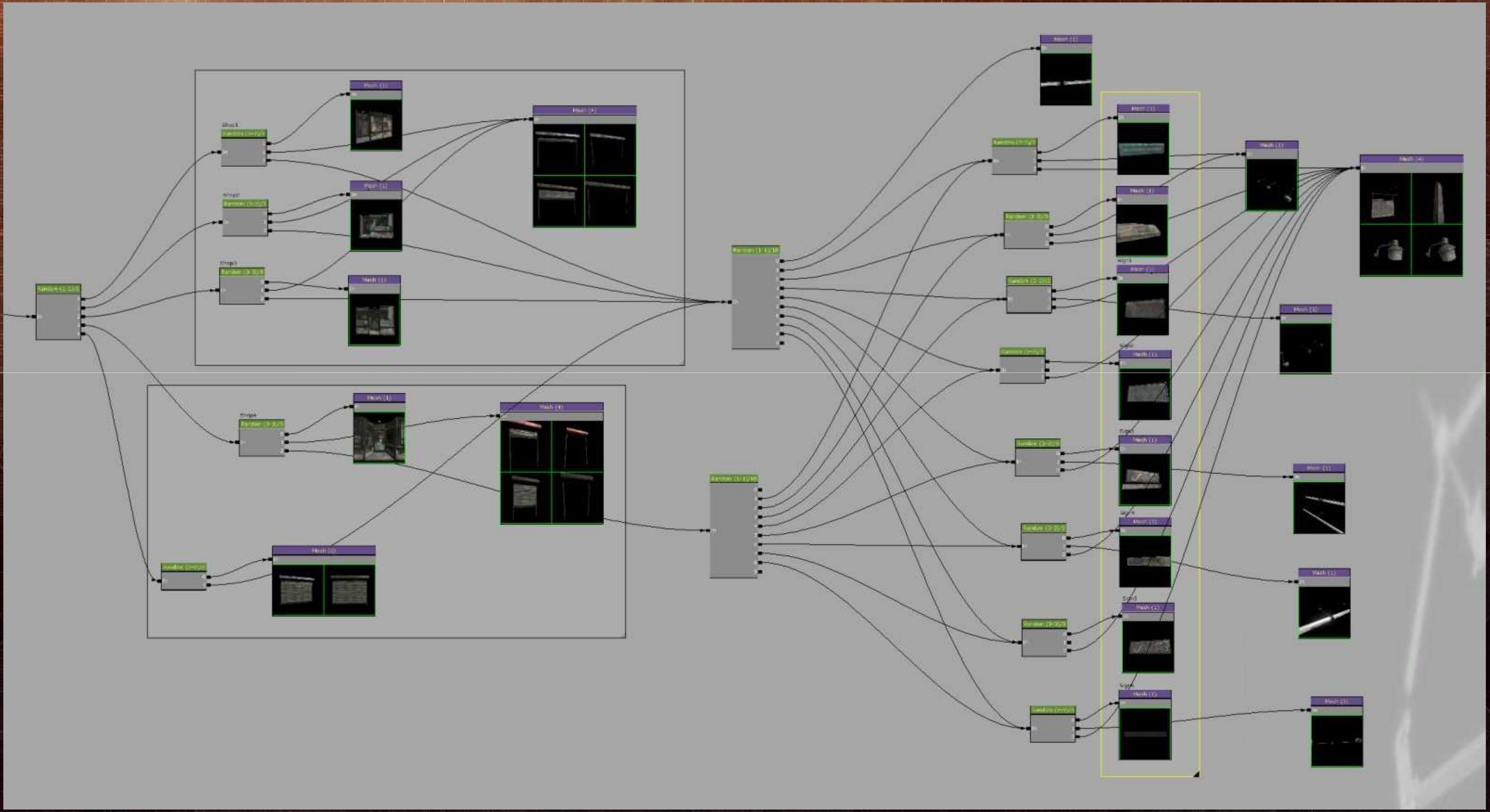
- Composite regions like random shopfronts



Random Rule

MIDTOWN MARKET
FERGUSON'S
Jewelers

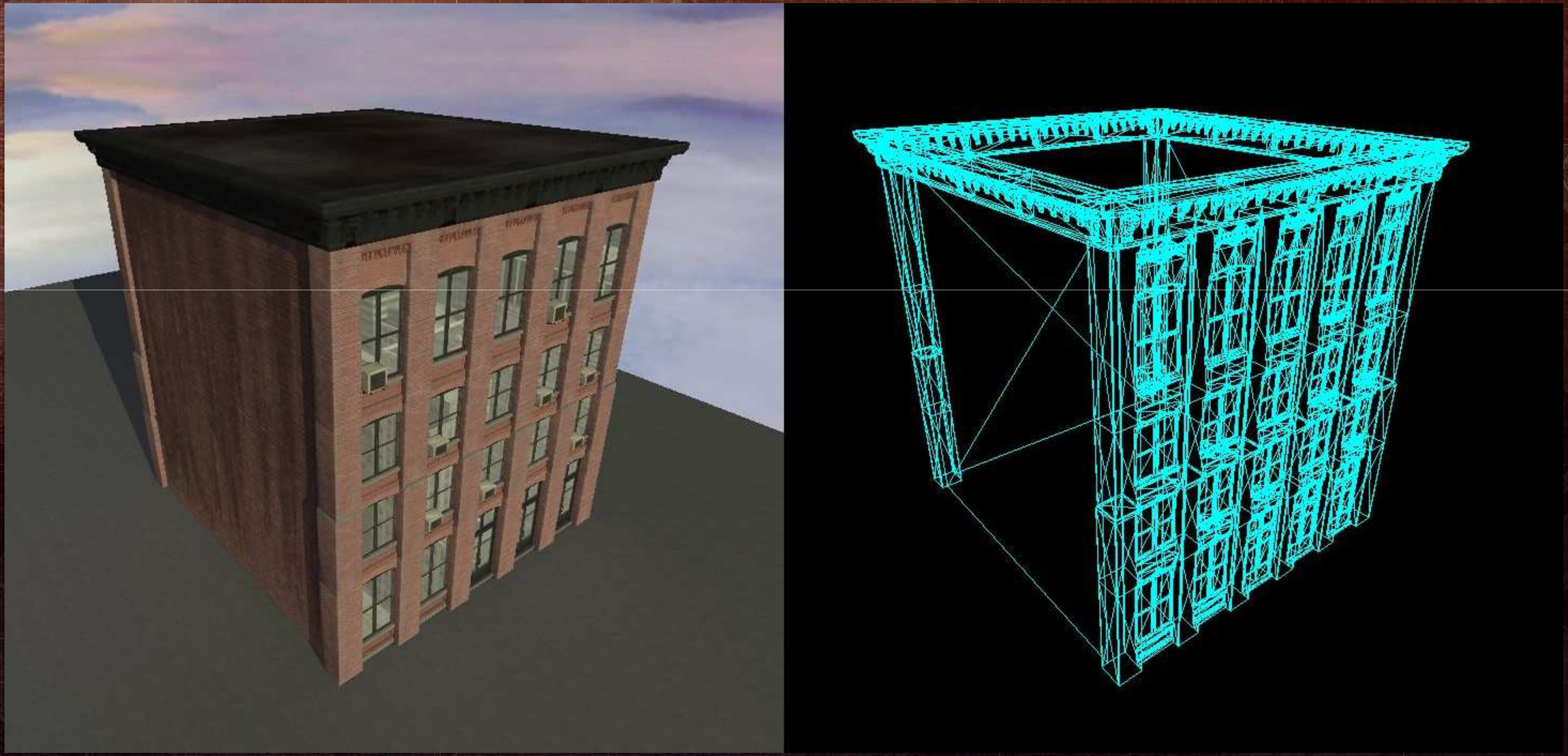
Random Rule



Quad Rule

- Sometimes you DO want a tiling material
- Simple variation of Mesh Rule
- Adjusts UVs to tile base on scope size
  - Same logic as Repeat Rule
- Second non-tiling UV channel

Quad Node



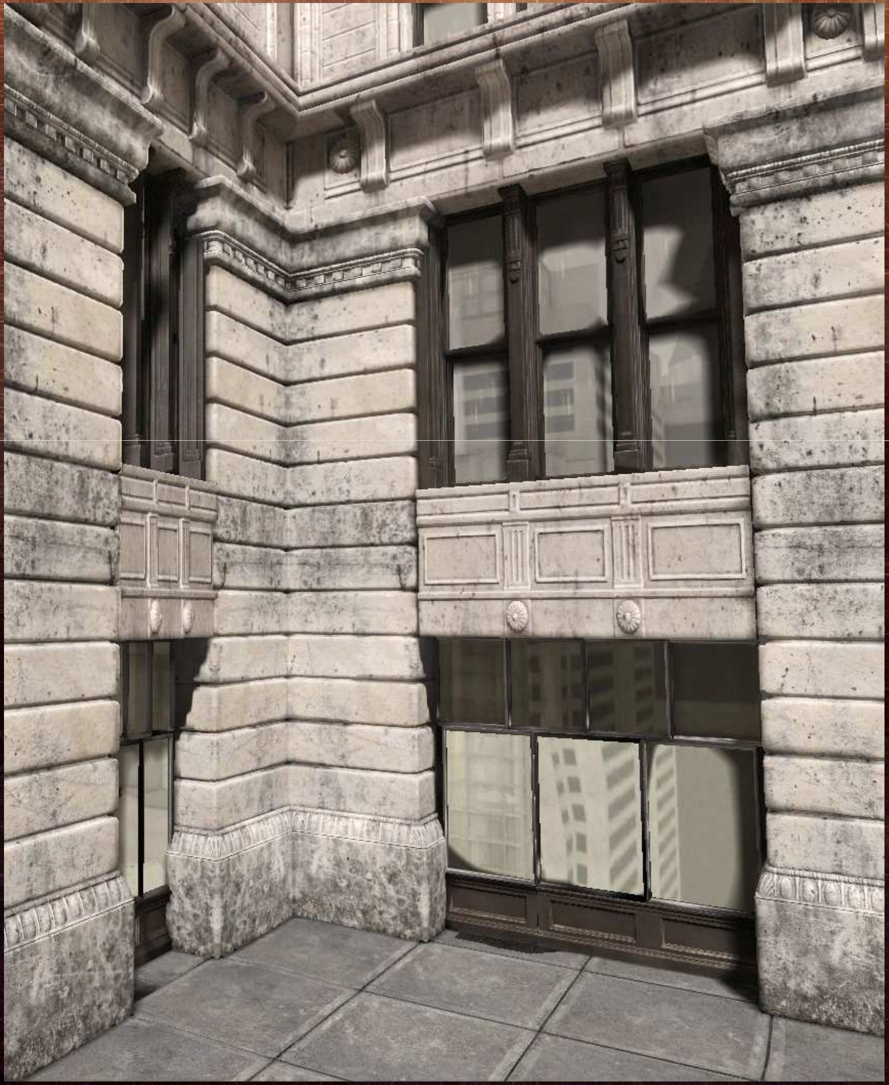
Sub Ruleset

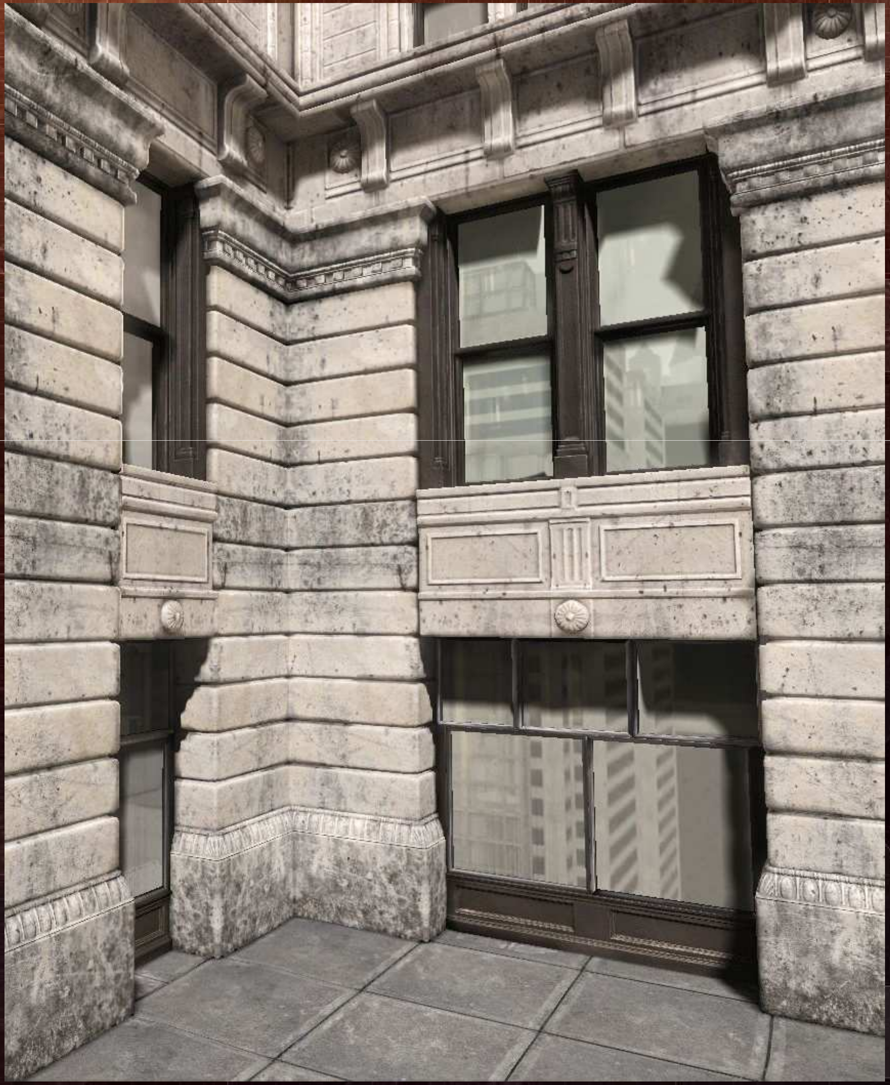
- Allow a Ruleset to refer to other Rulesets
- Complicated Rulesets can be reused.
- Terrifying prospect of recursive architecture!

Size Rule

- Simple choice based on dimension
  - Useful for fixing ‘squeezing’

Size Rule

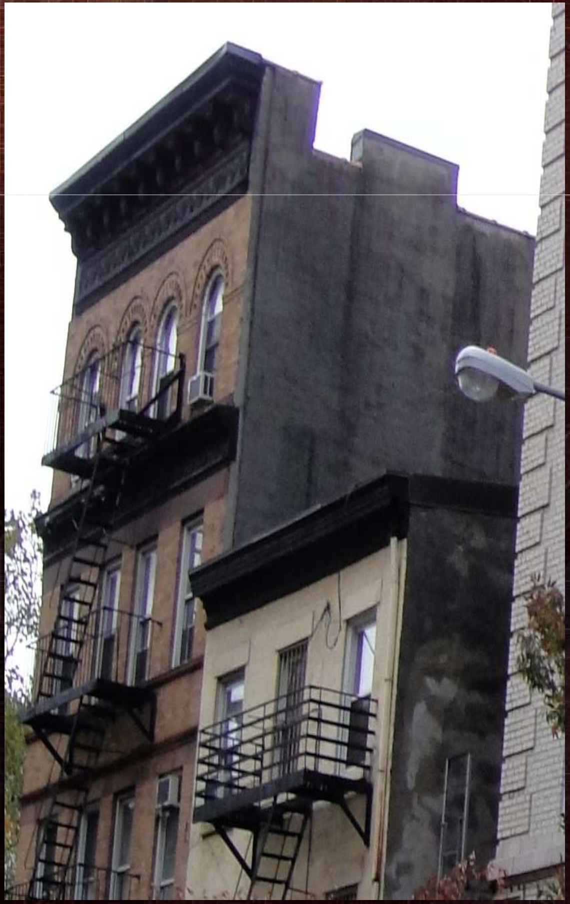

Without Size Rule


With Size Rule

Variations

Variations

- Each side of a building may need to look different



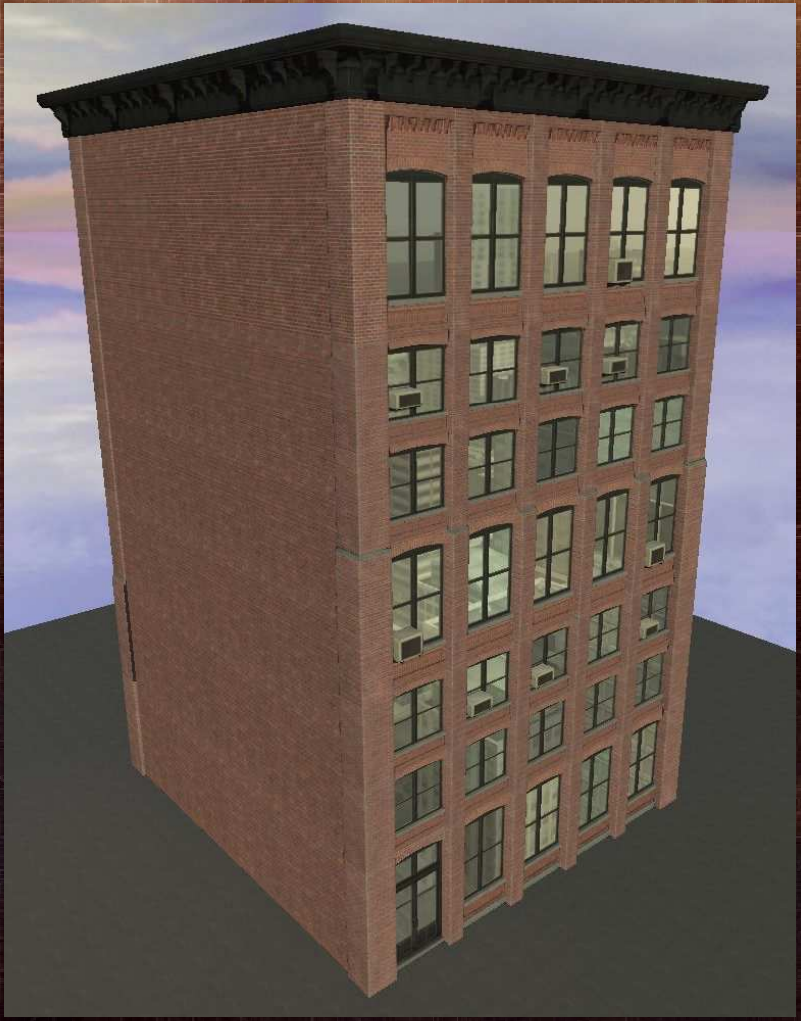
Variations

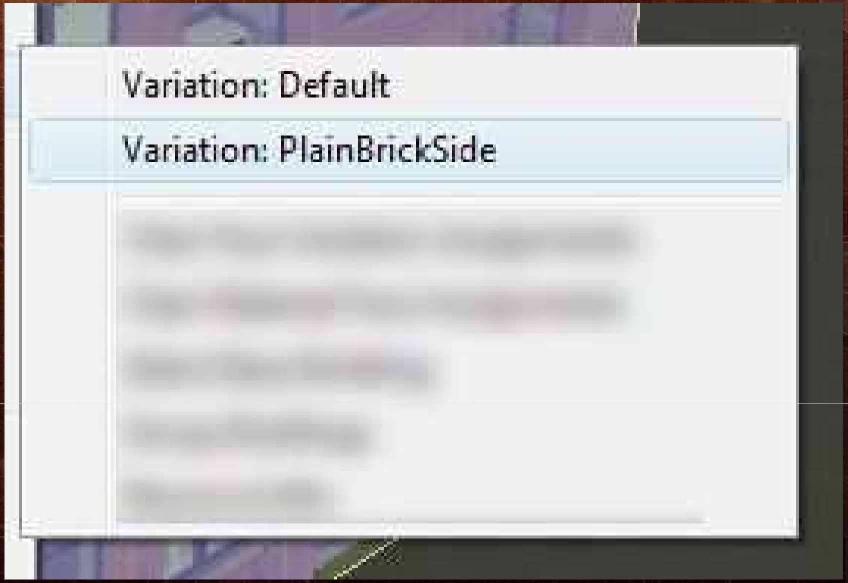
- Initially allowed Level Designers to assign rulesets per-face
  - Corners usually looked bad
  - Artists had no control over ruleset combinations

Variations

- Each face of a building volume can have a variation ‘name’ set on it
  – Special rule node uses that to decide which output to fire
  – Level designer can choose ‘front’, ‘side’ etc.
  – Ruleset designer can ensure they all match nicely

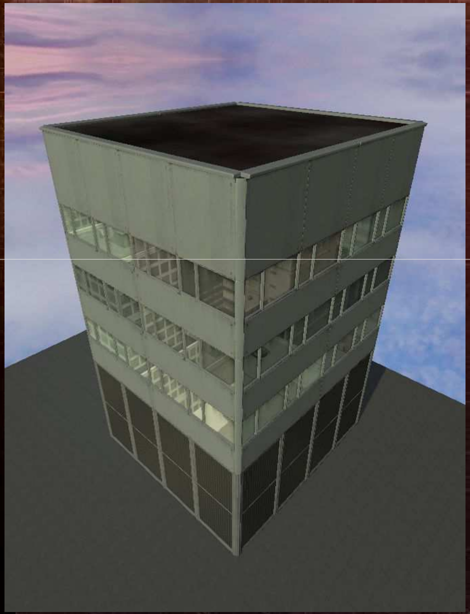
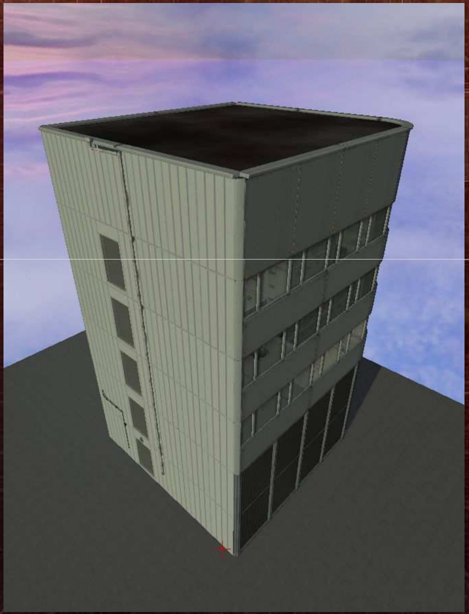
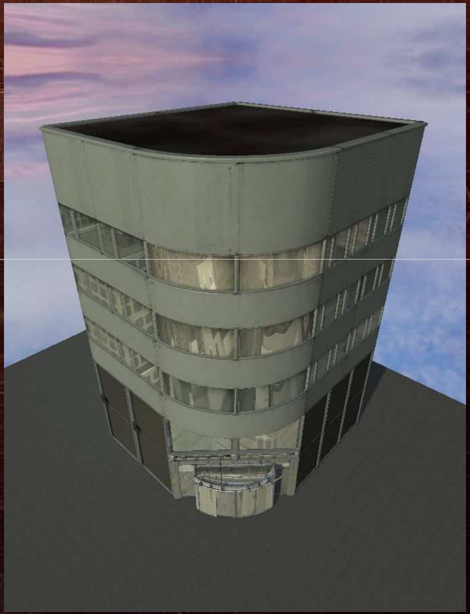
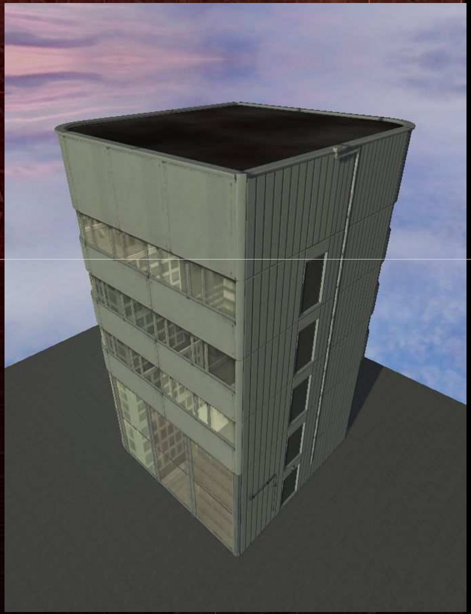
Variations





Context menu on each face offers variations created by artist

Variations

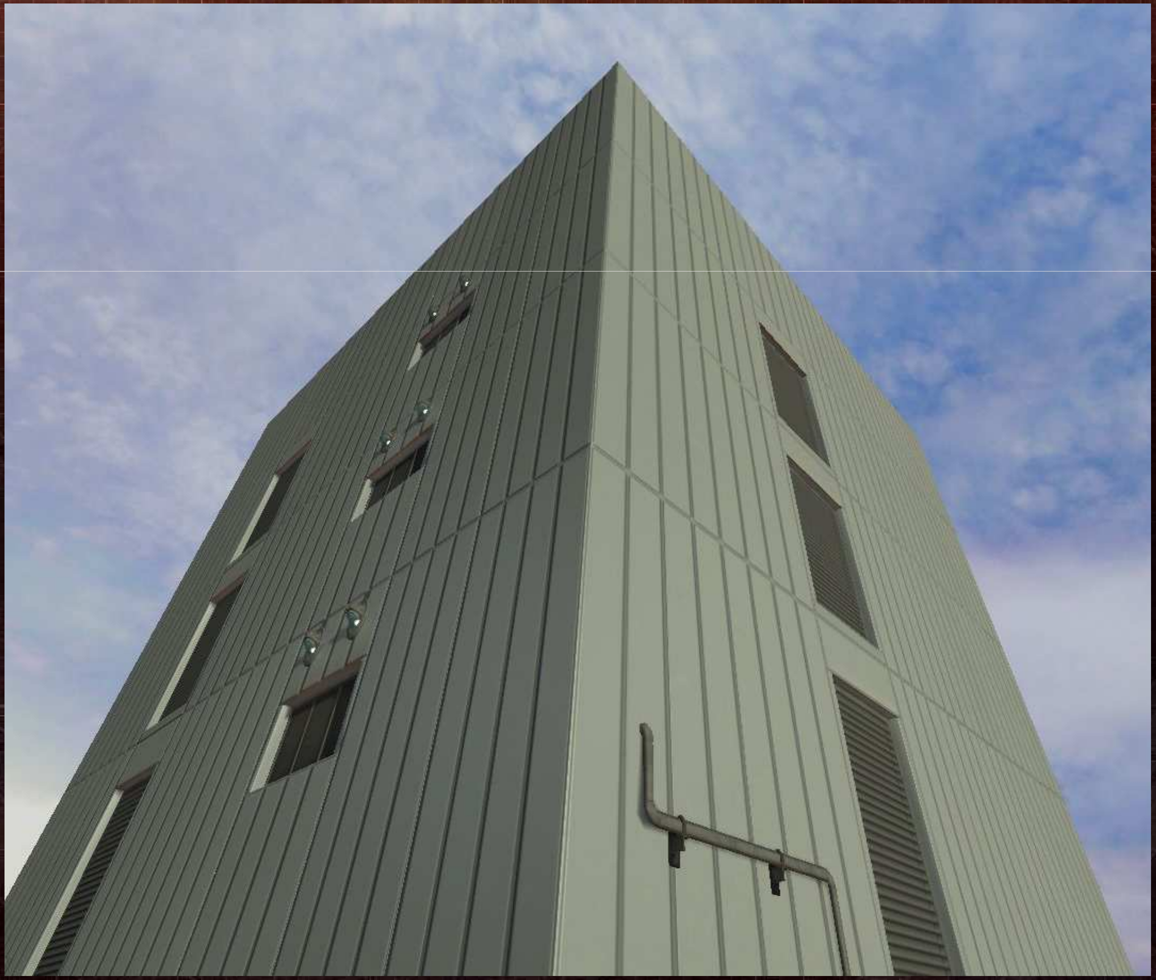
Corners

Trim And Corners

- Making trim work around corners is one of the hardest parts
- We came up with three approaches to decorating corners

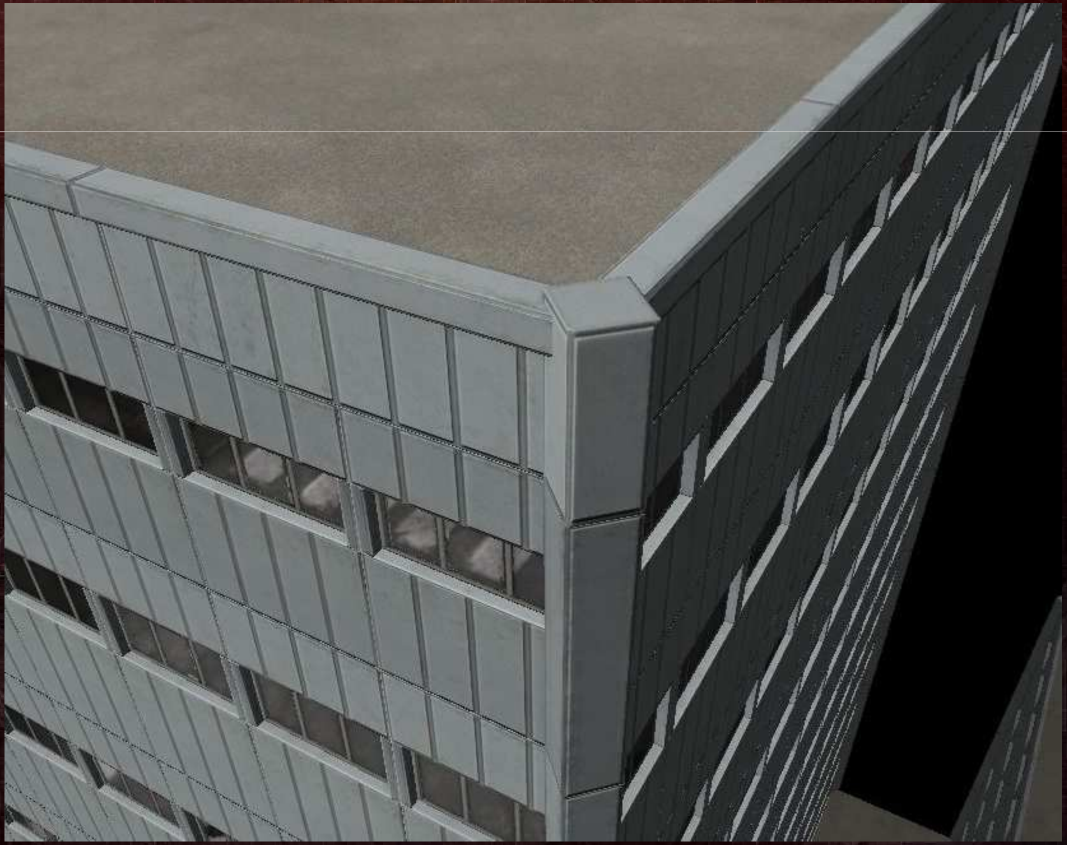
Trim And Corners

- Build Rulesets with flat edges
  - Only really possible with modern architecture



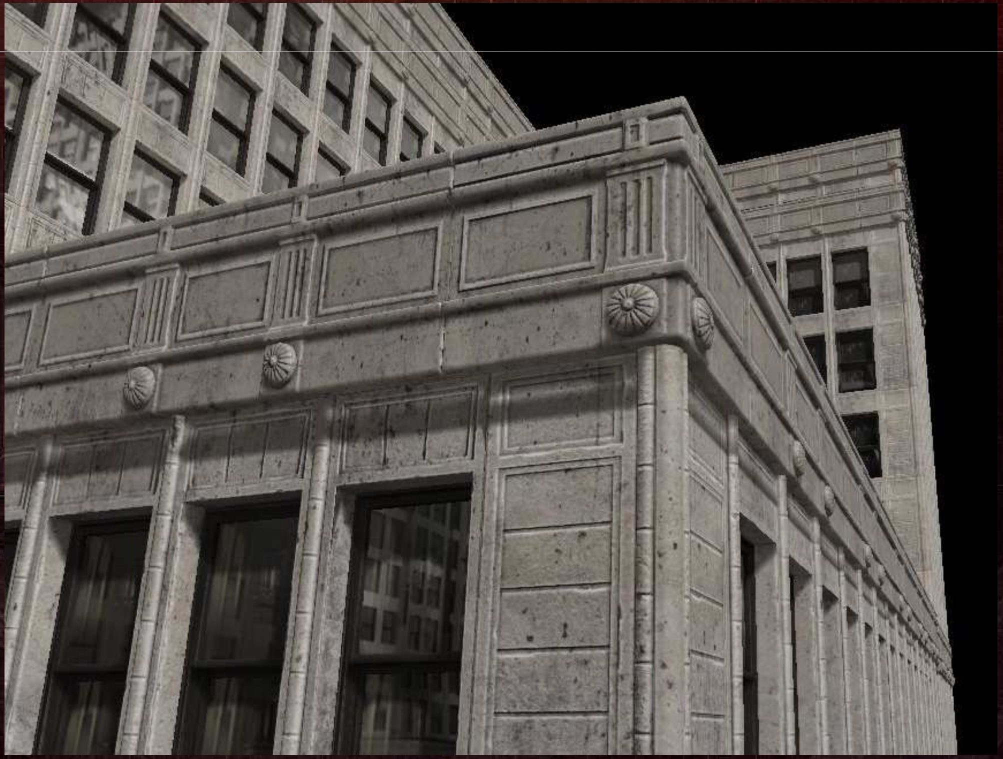
Trim And Corners

- Cover corner with mesh, at average angle between faces
  - LDs did this manually on previous games



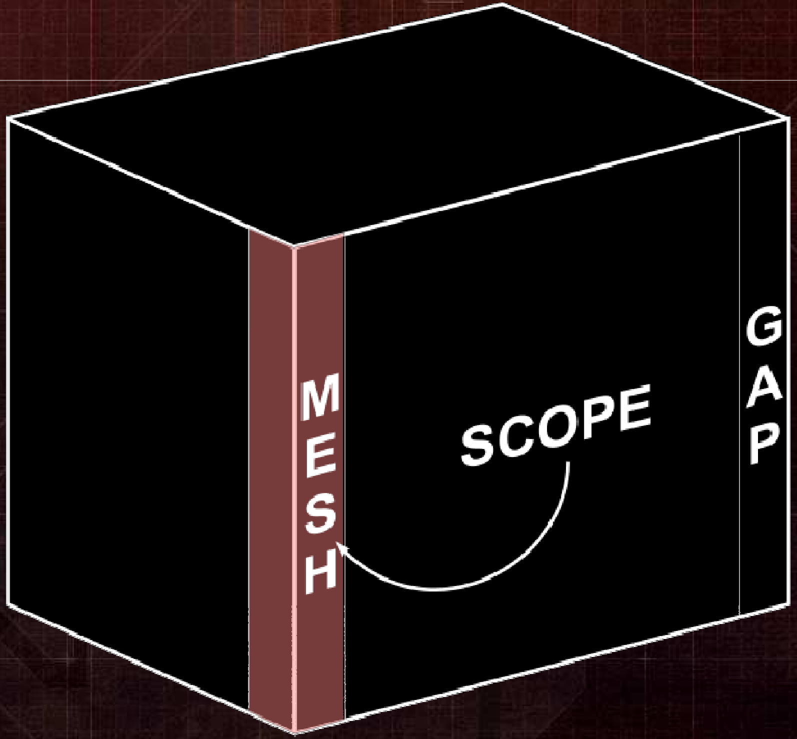
Trim And Corners

- Building custom corner pieces
  - use custom Rule to pick correct piece based on angle
  - lots of custom meshes need making
  - limit LD to certain angles



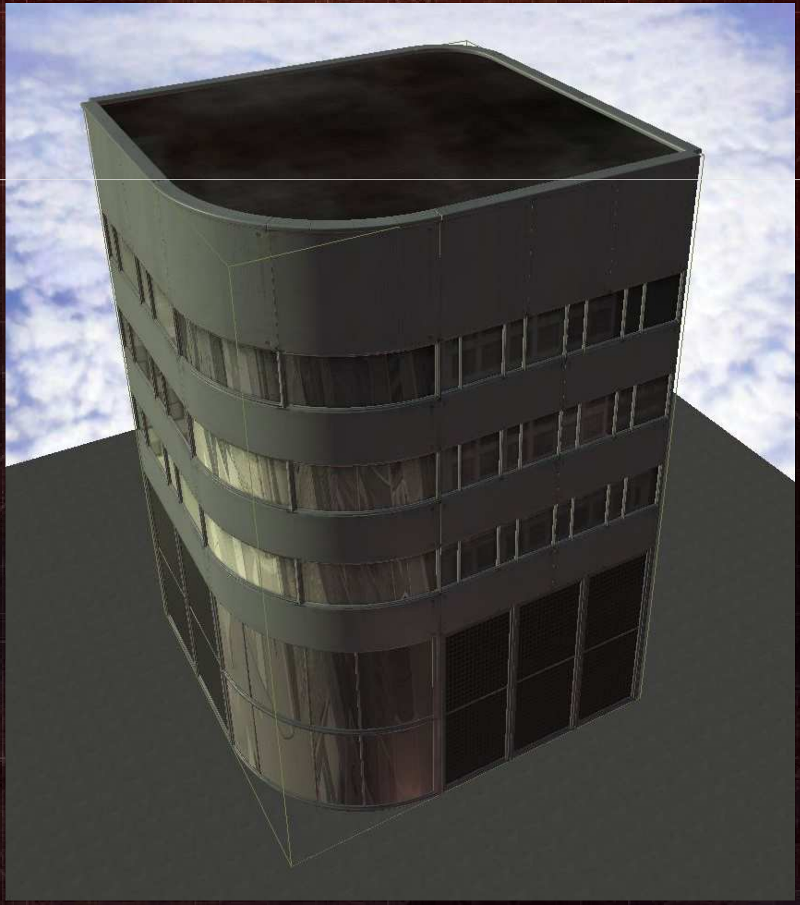
Corner Rule

- Each scope 'owns' its left edge
- Use angle to scope on left to pick mesh
- Asks scope to right how much 'space' to leave there



Corner Rule

- Allows mixing rule sets with different corner sizes

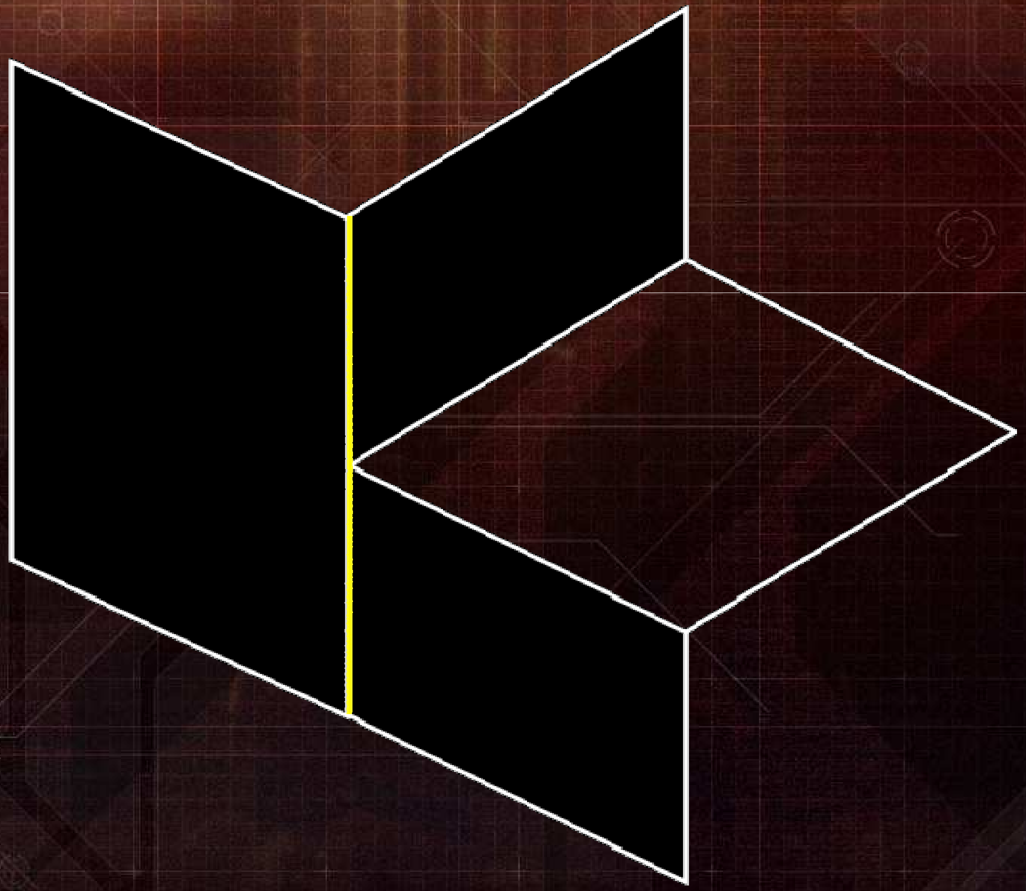


Corner Rule

- This requires edge<->scope map
  - Array of ‘top level scopes’
  - Array of ‘edges’
    - I am scope 3’s left edge and scope 12’s right edge
    - This is my start and end location
    - This is my angle
- Build this map as part of scope extraction

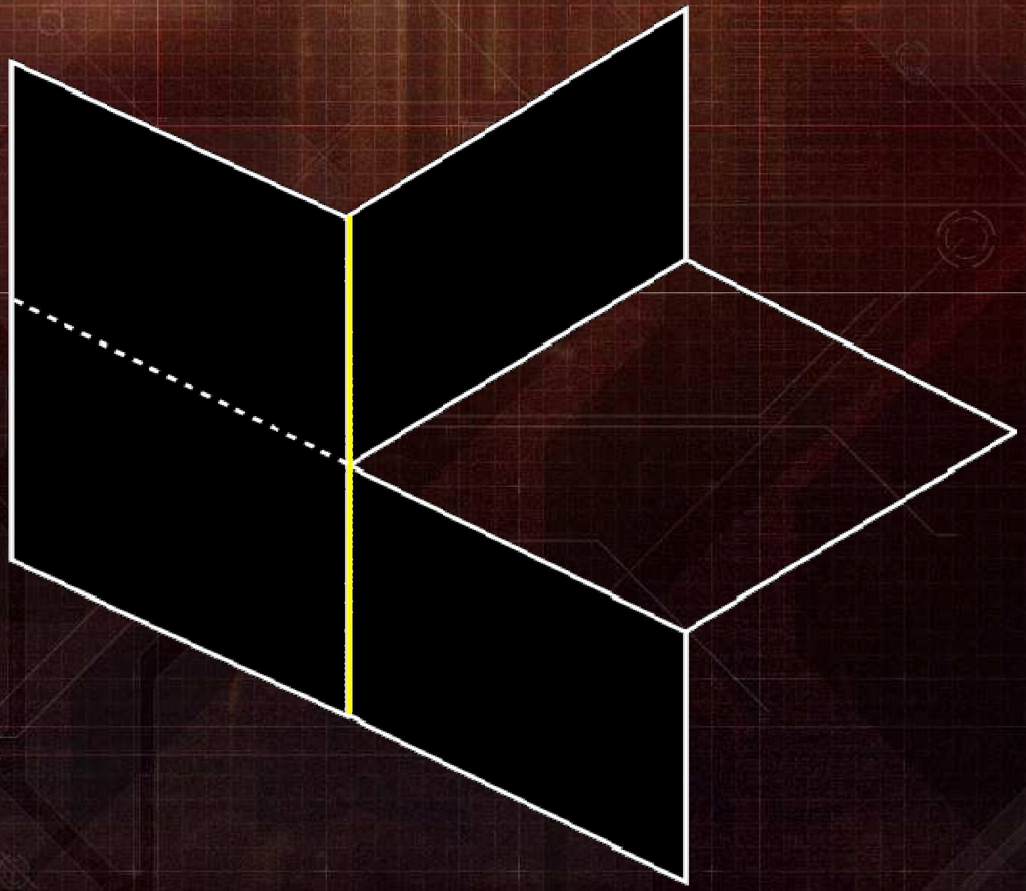
Corner Rule

- Requires each edge to only have 2 scopes



Corner Rule

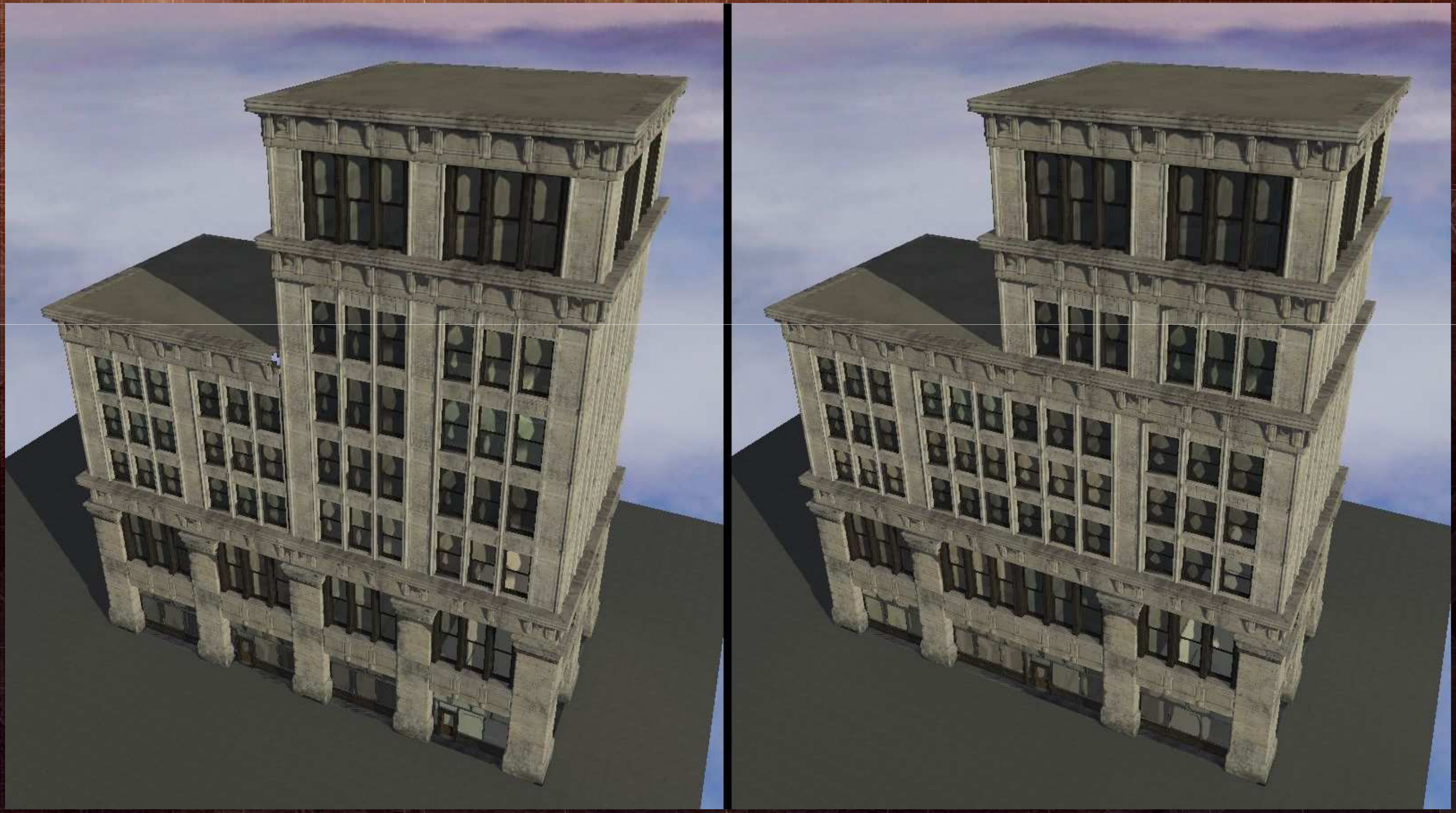
- Split entire building at each roof level



Corner Rule

- Also produces pleasing architecture

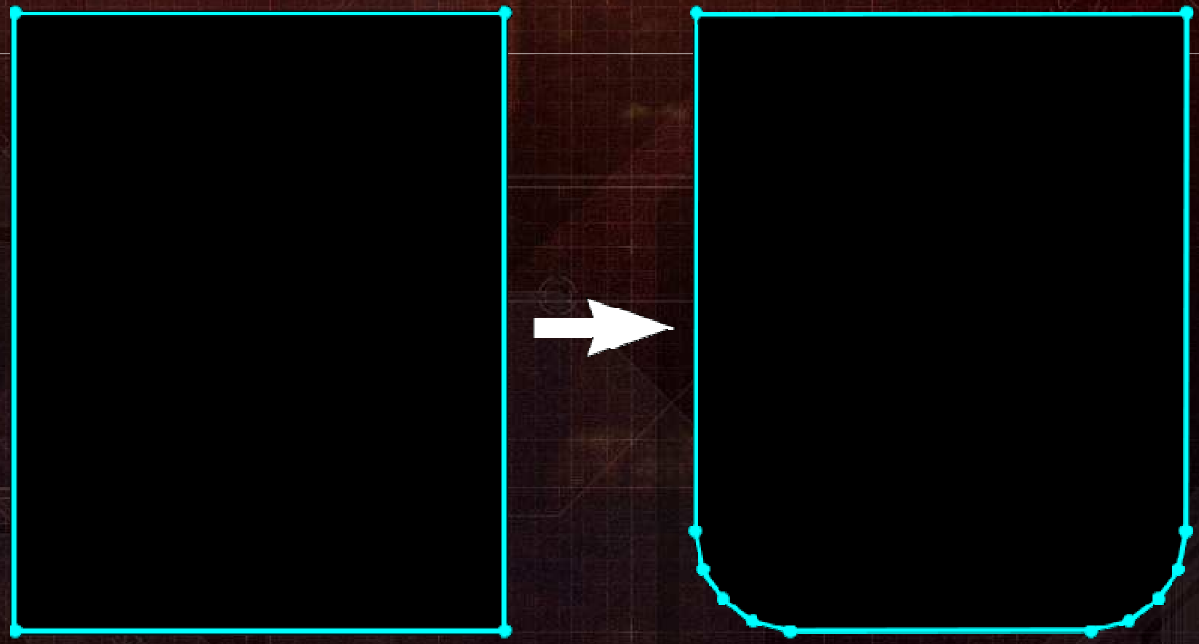
Corner Rule



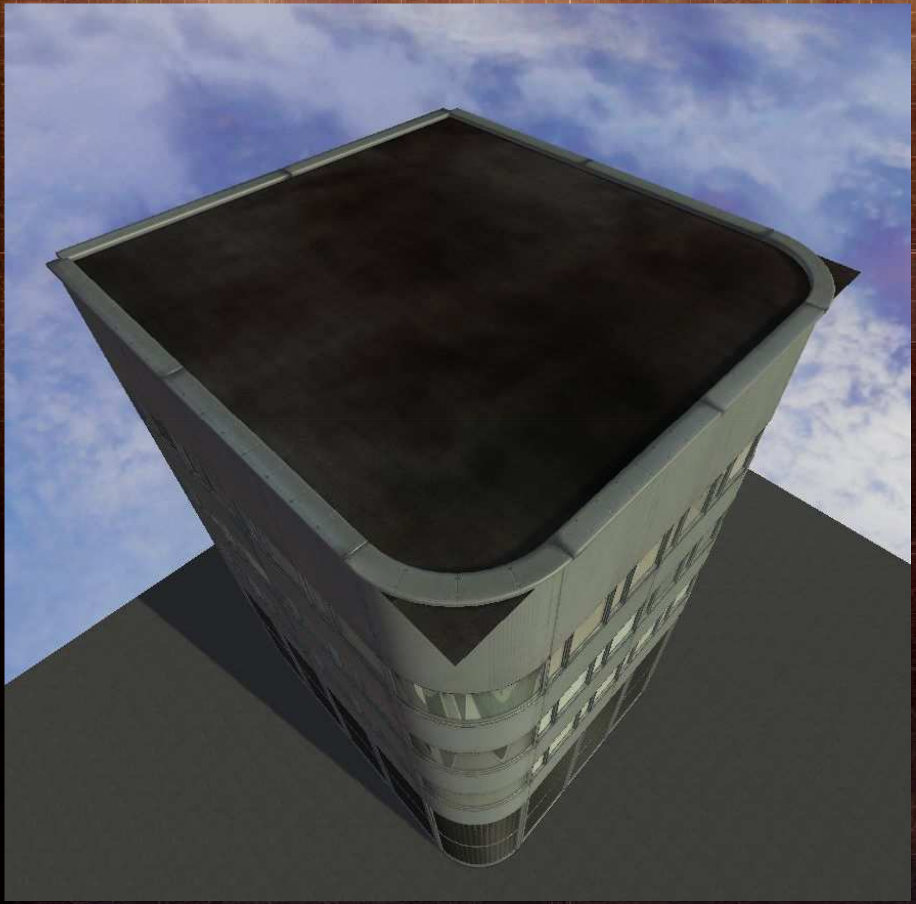
Other Features

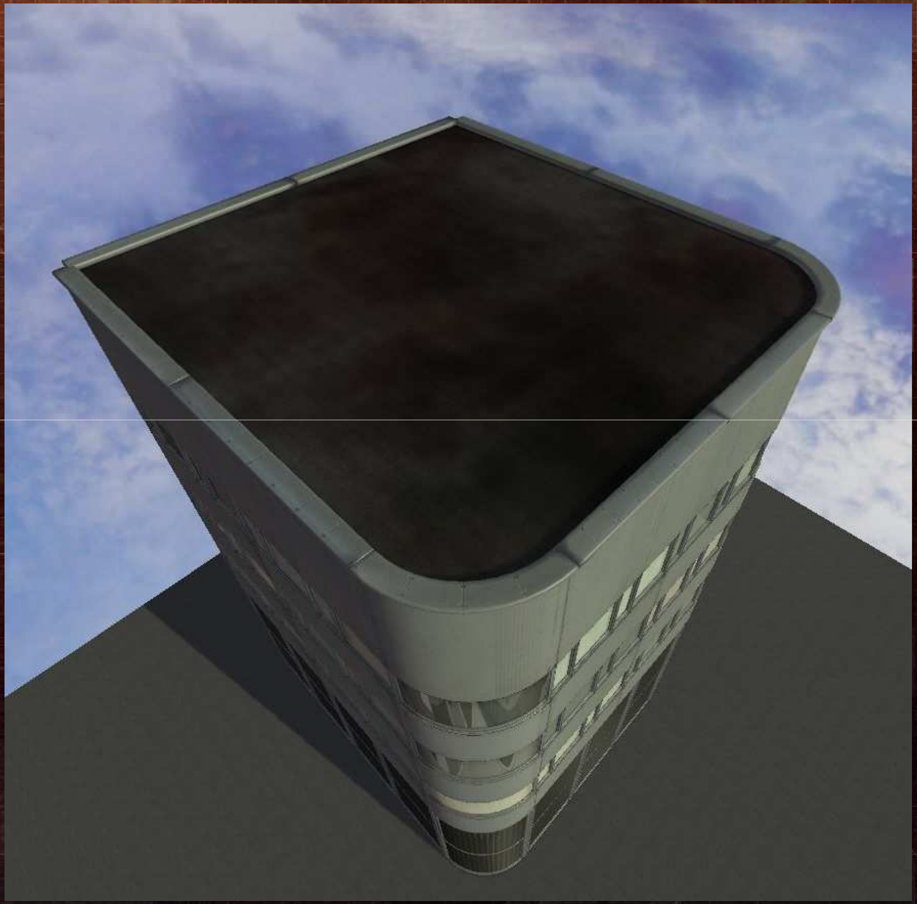
Roof

- Curved Corners
  - From each vertex of roof poly, can find its Corner Rule
  - Add options to Corner Rule to describe corner shape
  - Use that to reshape the corner



Roof





Floor

- Option to run rules ‘on top’ of big floor poly

Player Collision

- Can use the simple 'building volume’
  - Fast
  - Smooth
- Certain meshes can be flagged as having collision in addition to this

Parameters

- Expose parameters in shaders applied to building pieces
  - Wall Diffuse and Specular Color
  - Window Diffuse and Specular Color
- Gives more variation for no memory cost

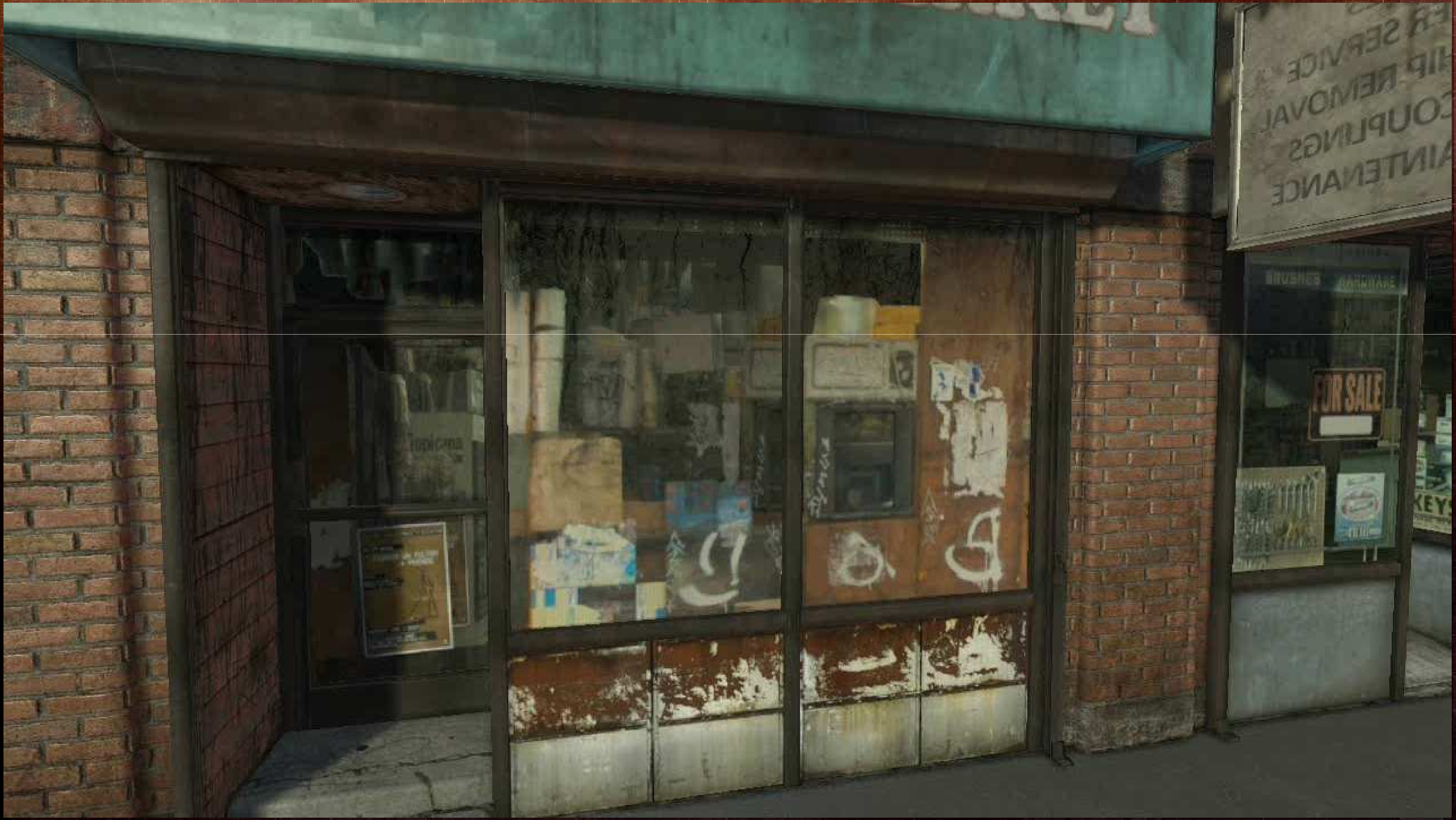
Rendering Approach

- Buildings tend to be made of many copies of a few meshes
  - trim, window frame, columns…
- Lots of draw calls
- Initially tried merging meshes to reduce sections
  - Huge vertex/index buffers!

Rendering Approach

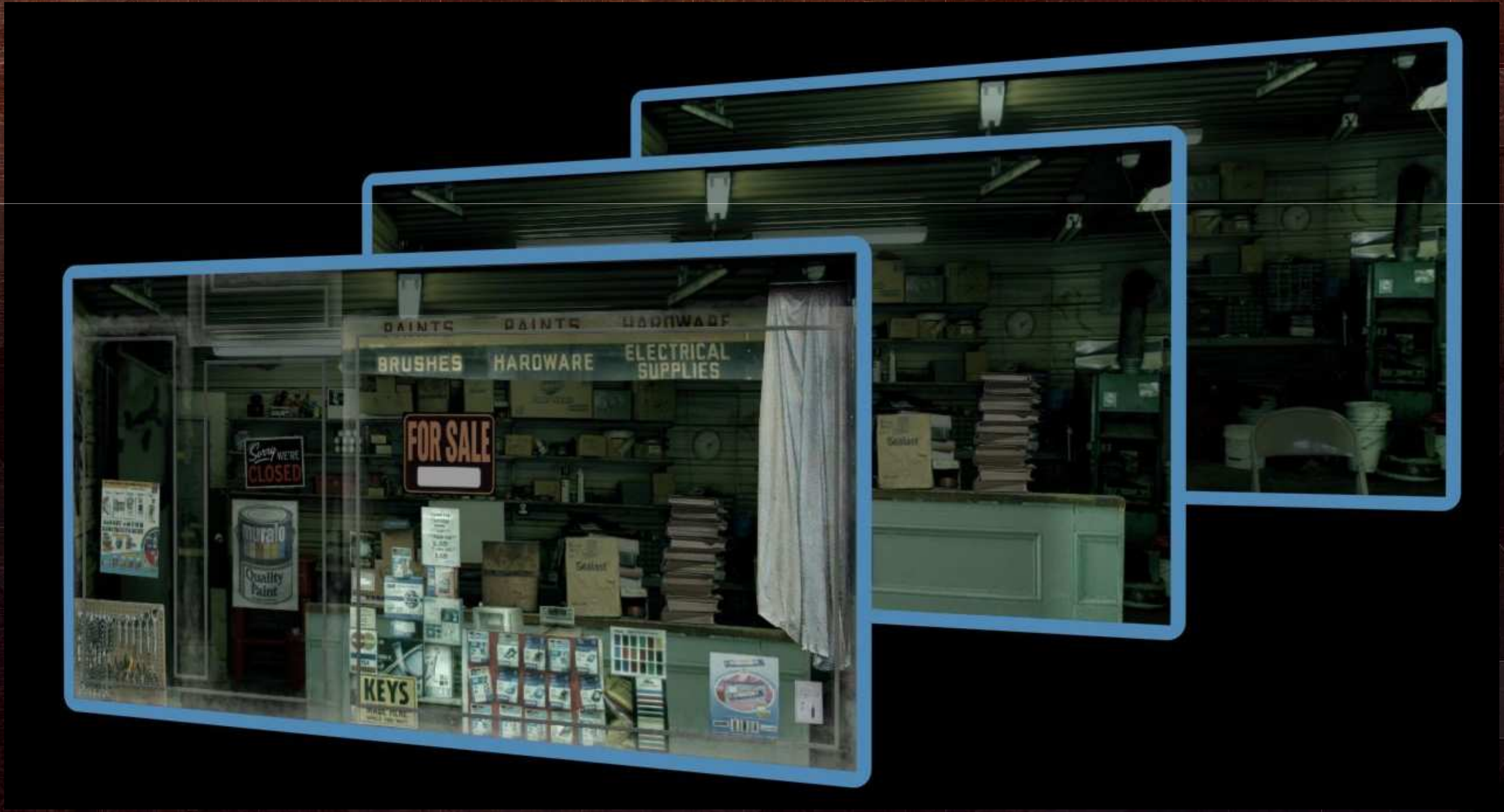
- Instanced rendering!
- Different on all 3 platforms
  - Need to duplicate index buffer etc.
- Trade-off index buffer memory for speed

Window Interiors



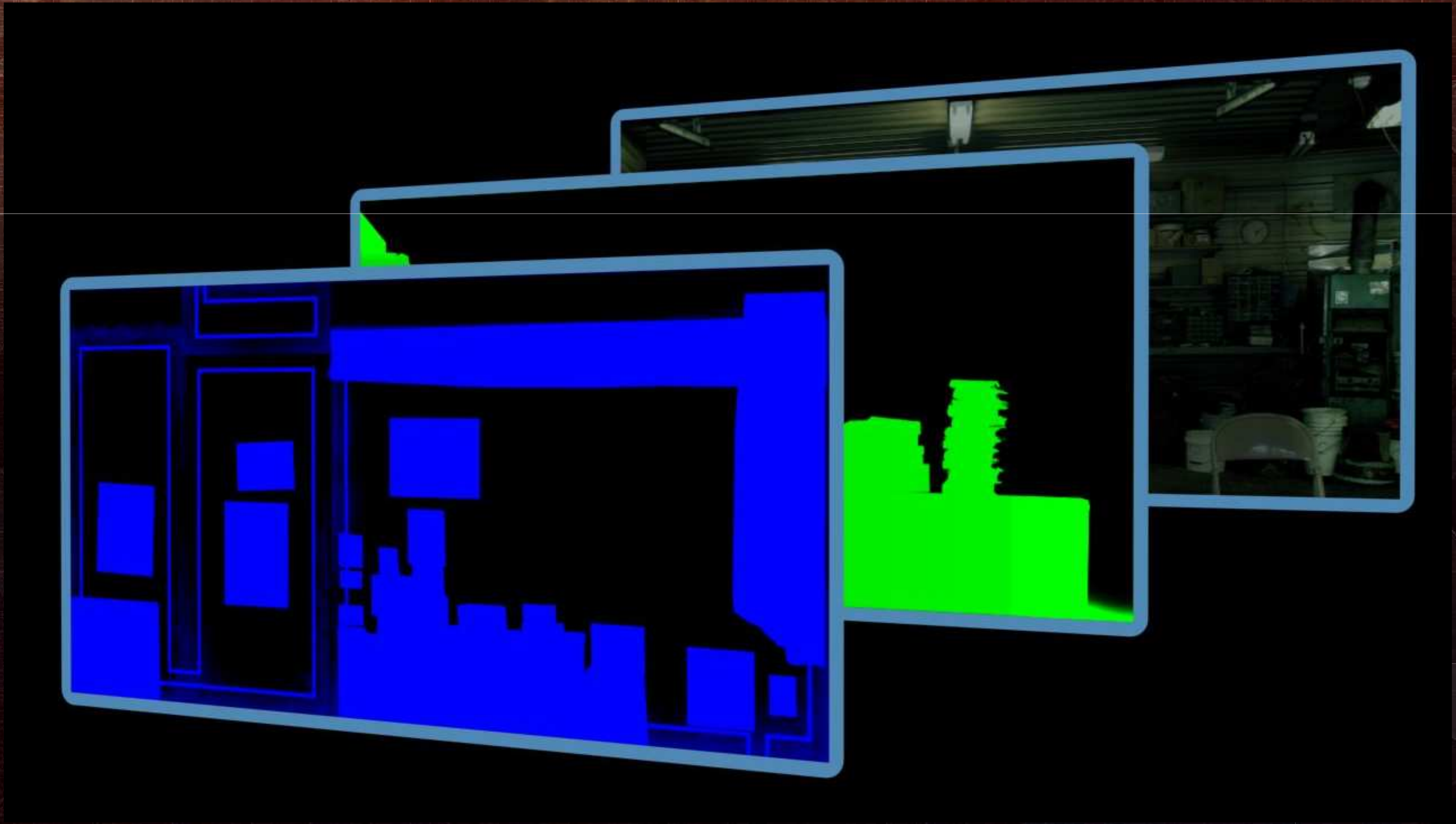
Window Interiors

- Three textures - one for each depth



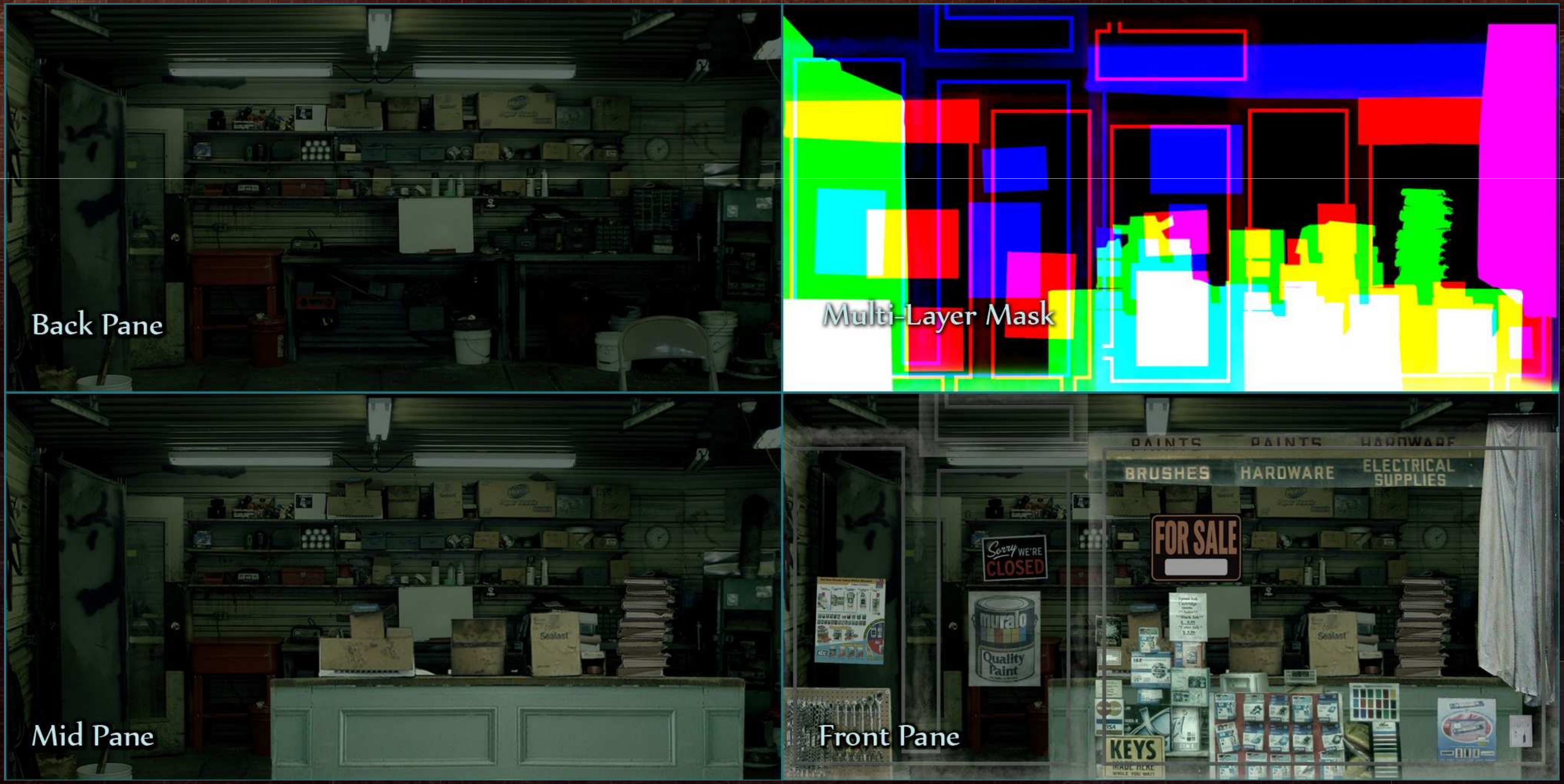
Window Interiors

- Use mask to offset texture UV based on cam vector



Window Interiors

- Pack all masks into one texture

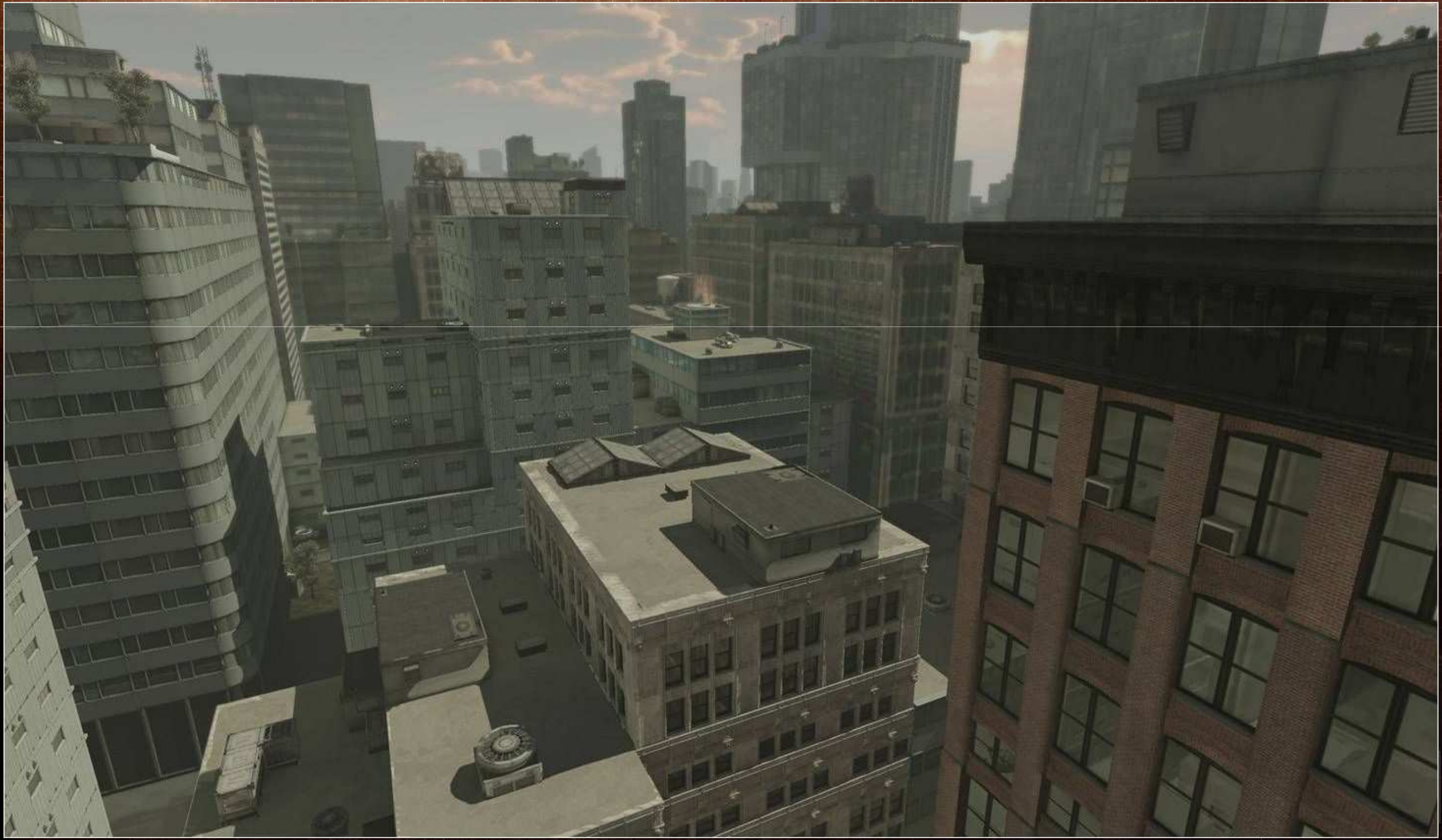


LOD

Building LOD

- Low LOD mesh is automatically created
  - Mesh the same shape as the volume used to construct building
- Renders high detail meshes into texture
  - All faces atlassed into one texture
- Low LOD mesh is always loaded
  - Textures can stream in different mip levels

Building LOD

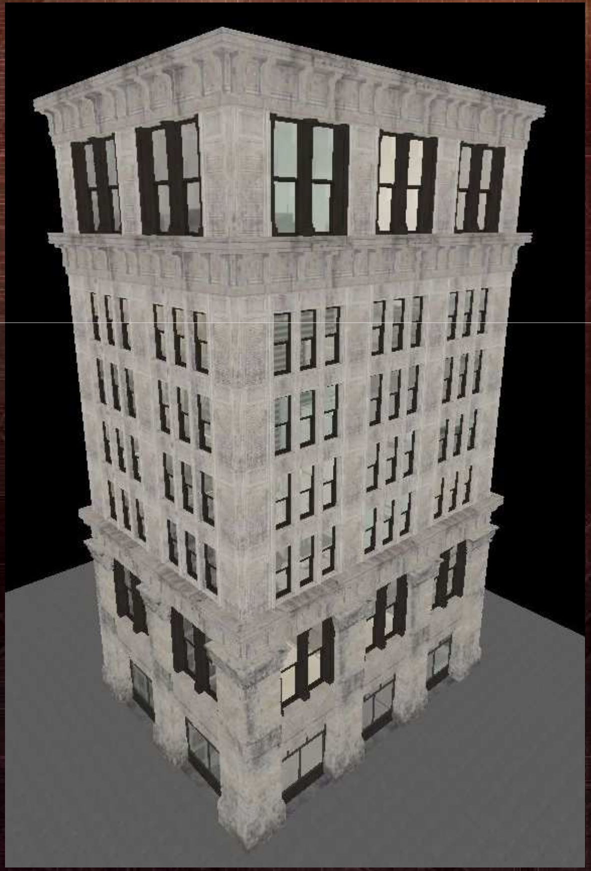


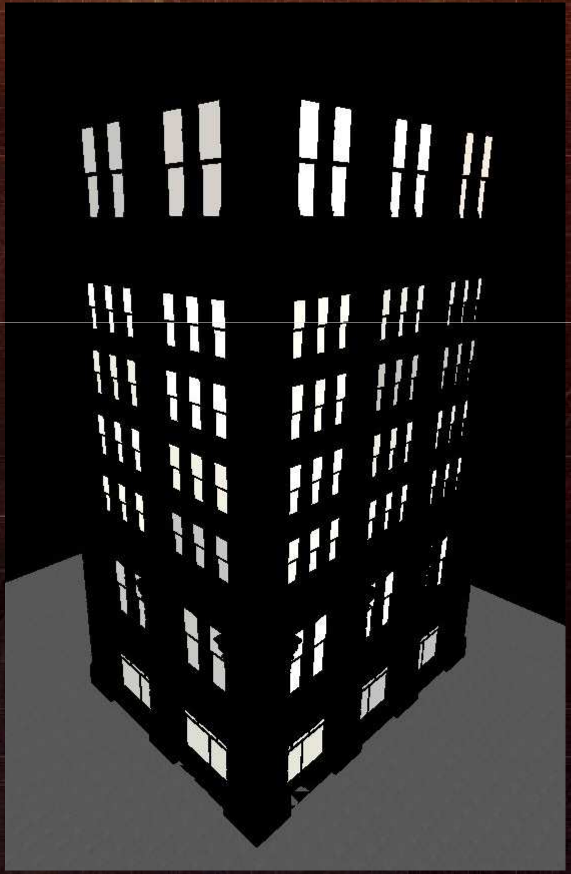
Windows

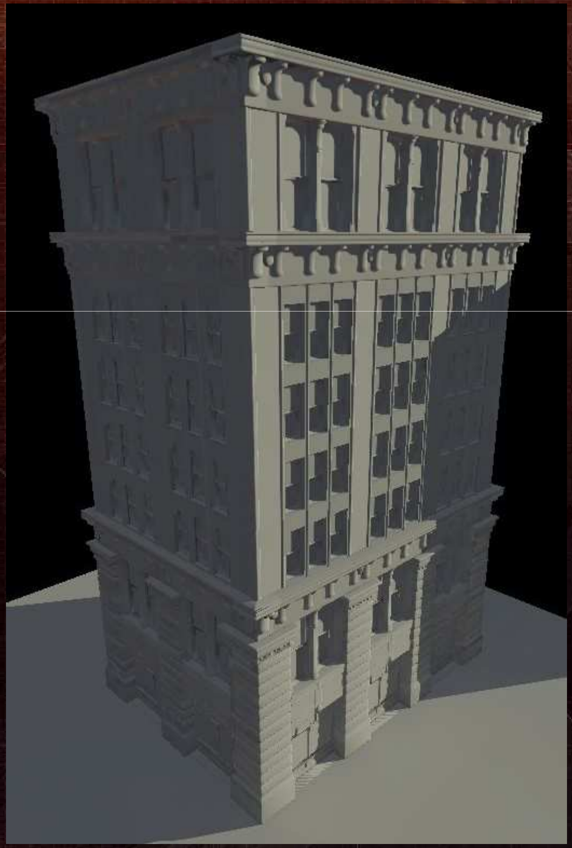
- Buildings all have glass windows
  - – reflective with cubemaps
  - – LOD needs a mask for reflective windows
  - – losing reflection creates very noticeable pop

Windows

We render 3 passes:

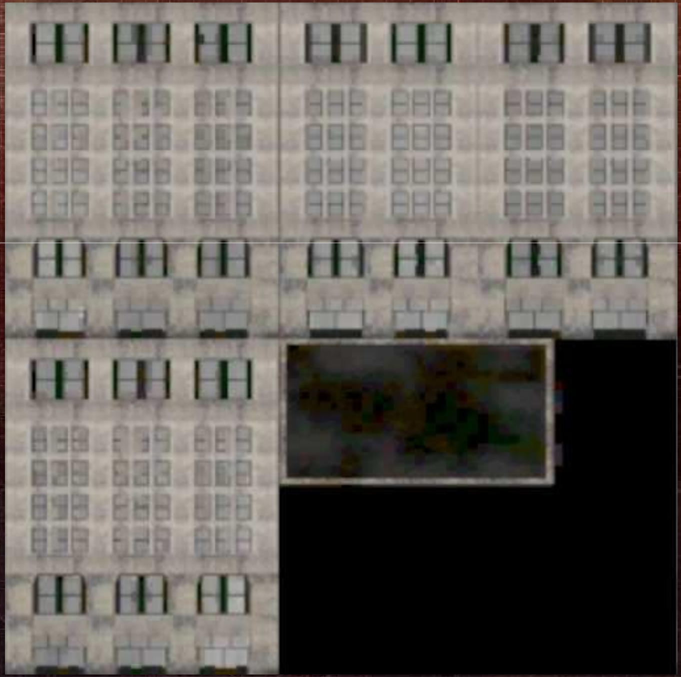

Diffuse


Window Mask

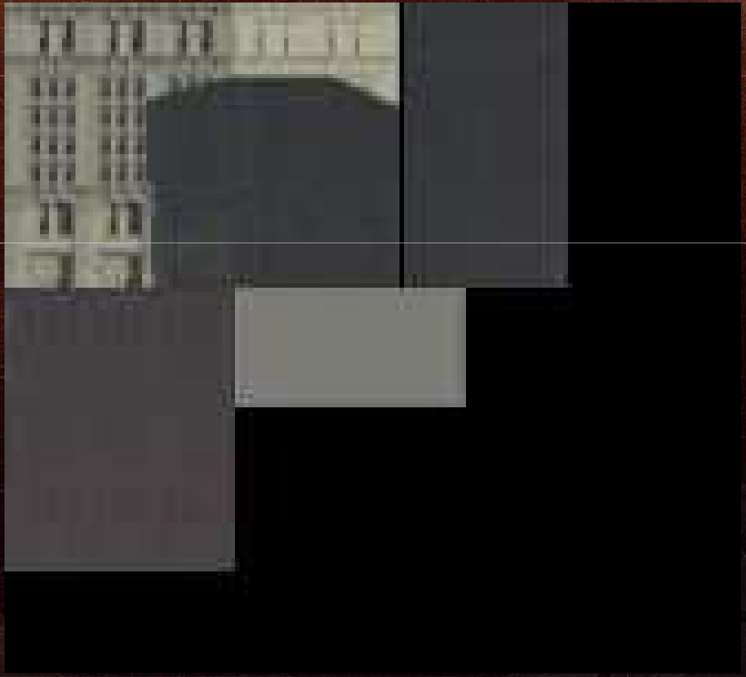

Lighting

Windows

Diffuse + Window Mask
- Mask stored as 1 bit alpha in DXT1



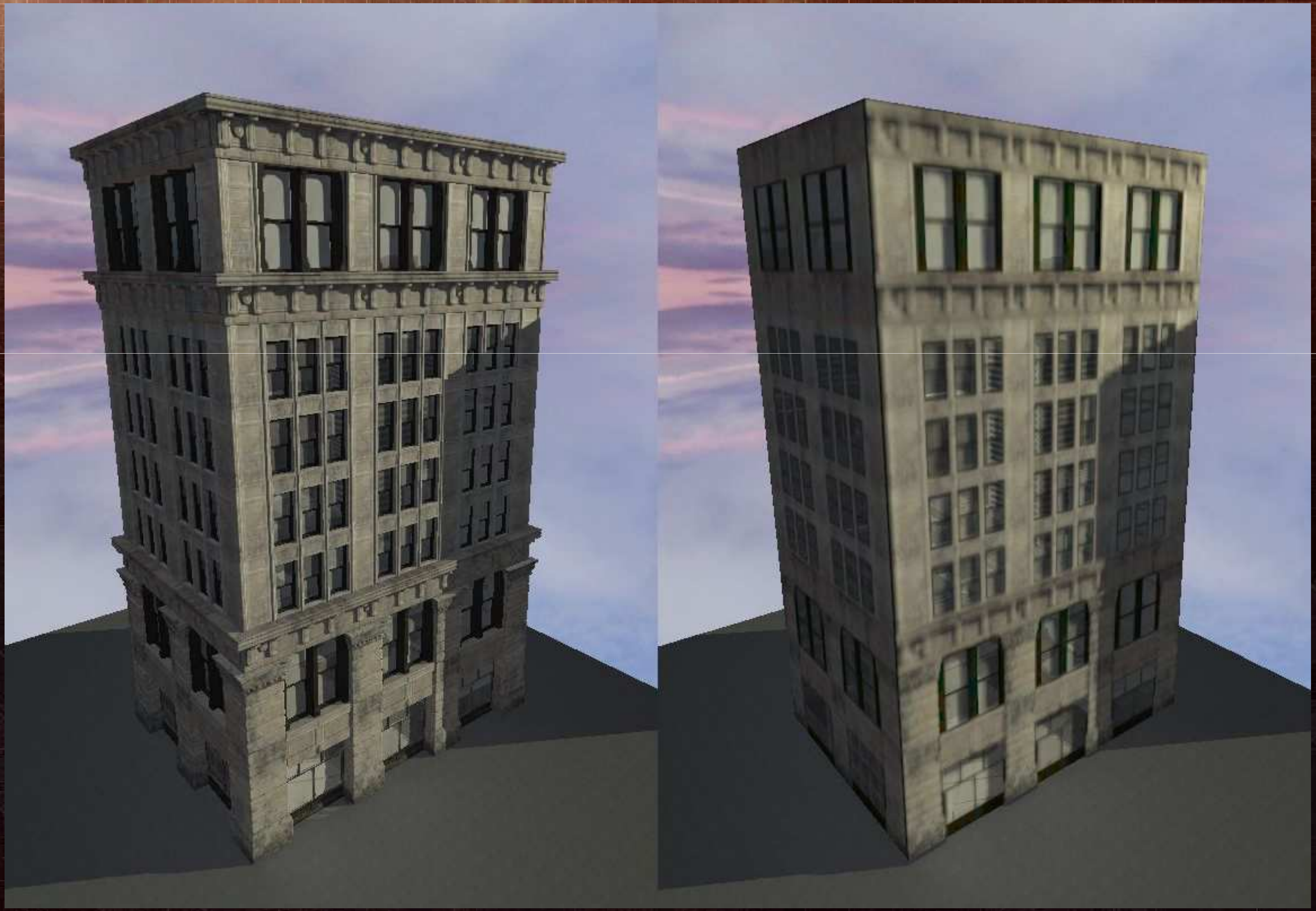
Lighting
- Lower resolution



Windows

- Advantages to 2 texture approach:
  - Lighting texture can be lower resolution
  - Lets us light cubemap ‘glass’ areas
  - Allows building instances
    - Share Diffuse but unique Lighting
    - Reduces variety of buildings, but uses less LOD texture memory

Building LOD

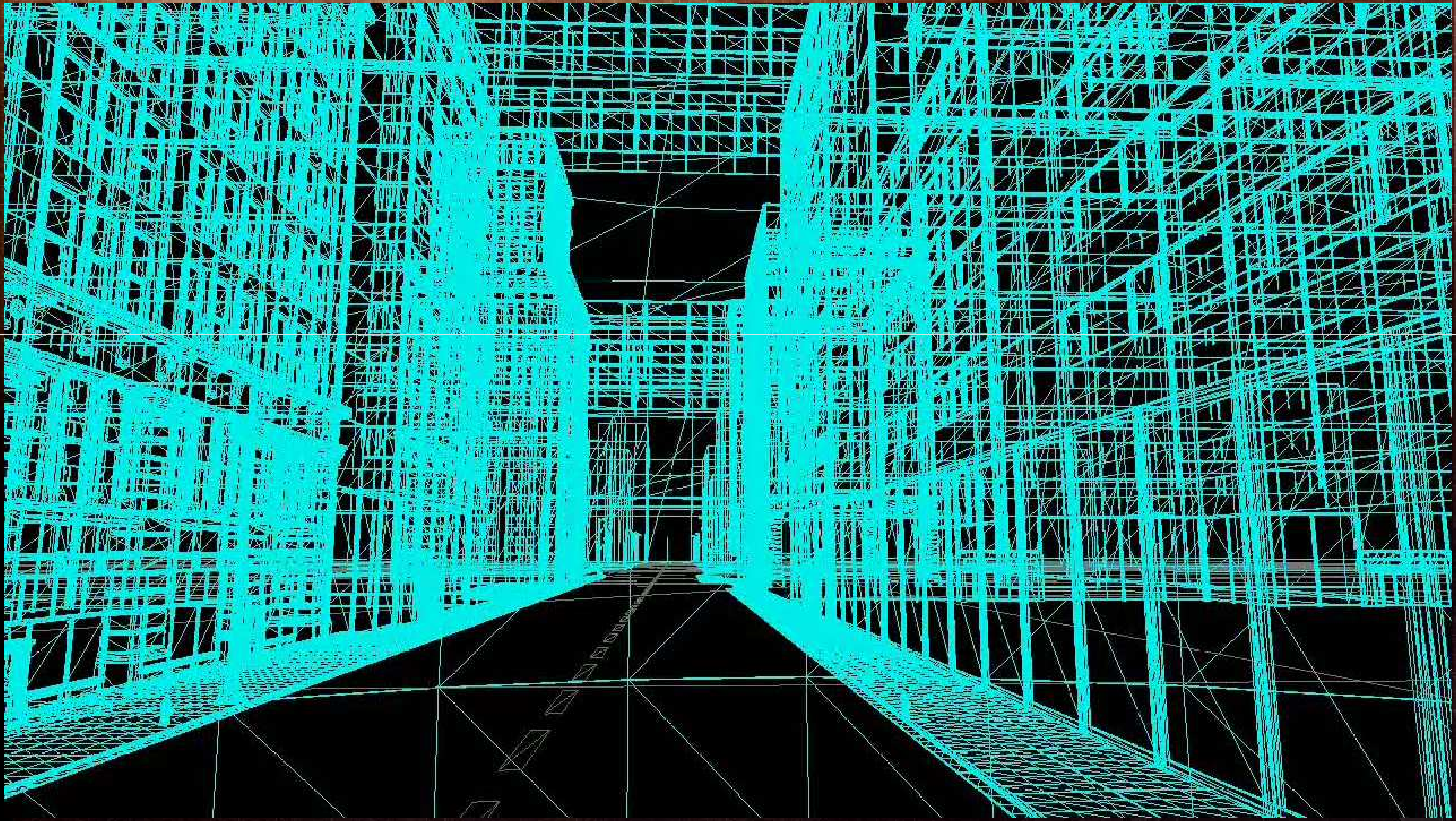


Transitions

- Using dither approach to transition
- Objects attached to building change when building does
  - A/C units, pipes, signs etc
  - Captured by render-to-texture process

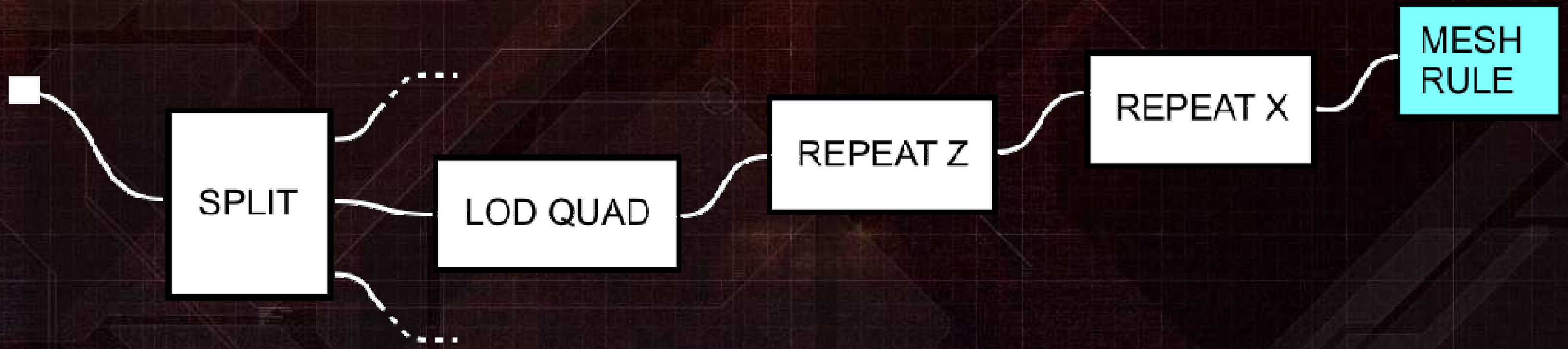
Transitions

Transitions

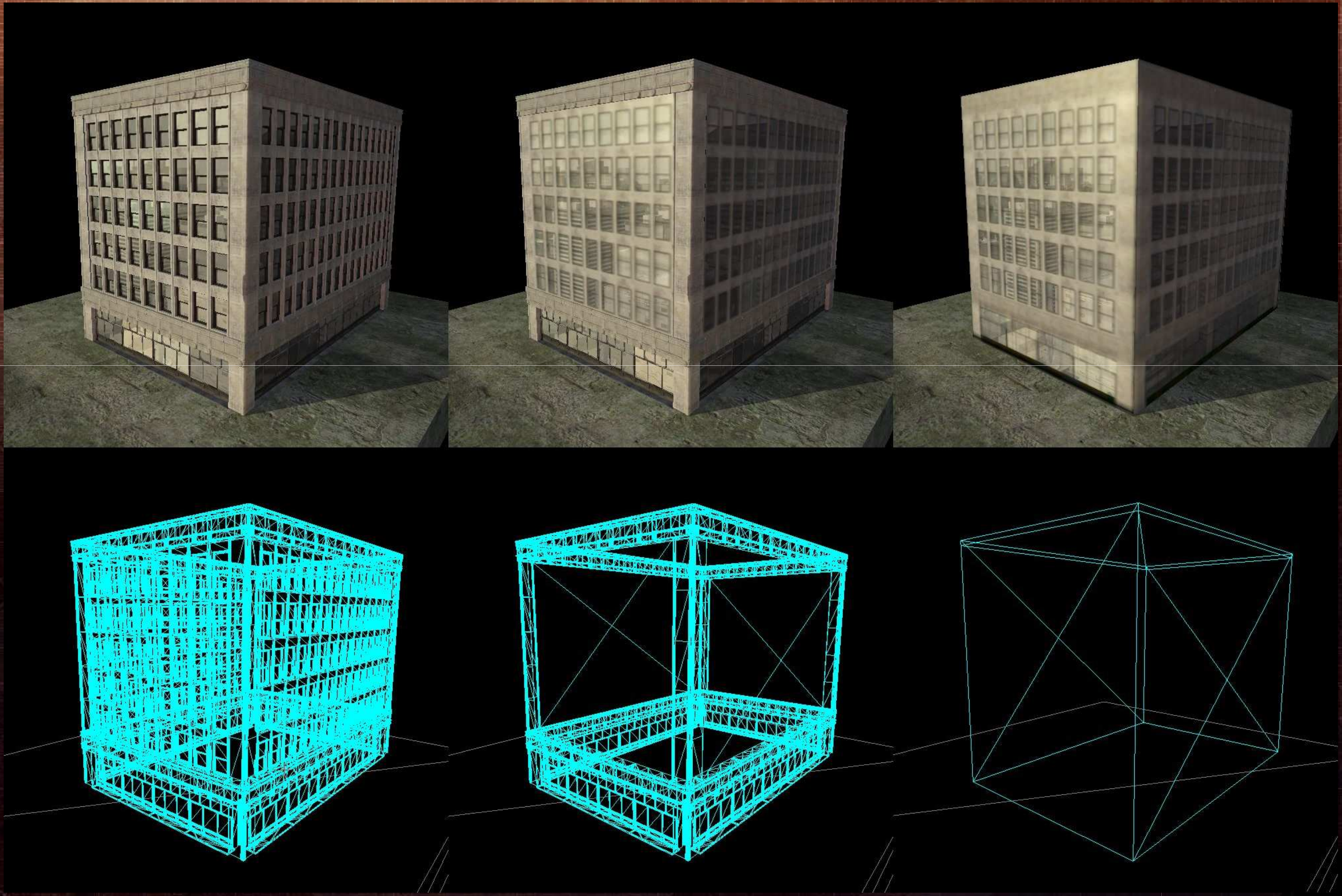


LOD Quad

- Idea for automatic intermediate LOD
- All meshes after rule collapse to single quad
- Quad uses LOD texture



LOD Quad



What’s Next

What’s Next

- More orthogonal variation ‘channels’
- Decal support within ruleset
- Apply rules to triangular regions
- Interiors
  - Transition (doors)
  - Collision

Conclusion

Conclusion

- Good looking buildings with high visual density
- Easily change shape and size for gameplay
- Automatically generate LODs

Conclusion

- Keeps artist and level designer workflows from colliding
- One change to a ruleset, the whole city changes
- Artists willing to learn a crazy new system and push it are invaluable and awesome
  - Thanks Pete ☺

Conclusion

- Rulesets everywhere!

Questions?
www.epicgames.com
www.unrealtechnology.com
Booth: BS200 South Hall