

Procedural City Detailing



Alex Mole

Motivation

- A large amount of game budget [time & money] is spent on content creation
- A substantial amount of this work is repetitive
 - particularly in environment creation, and especially if the environment is a city
- Computers are good at repetitive tasks!

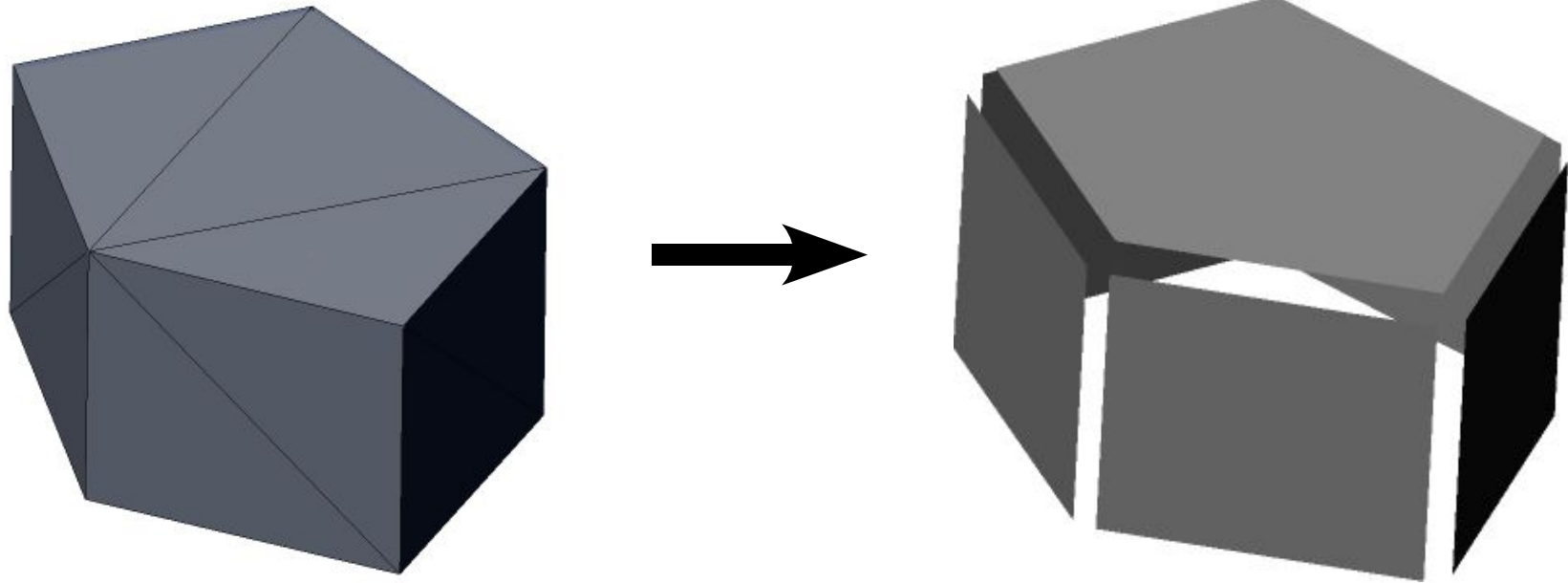
Existing work

- Photogrammetry
 - finding building outlines from photos and generating geometry from them
- Custom tools
 - deal with geometry at the “room”, “window”, “door” level rather than the polygon level
- Complete procedural generation
 - generate entire city from very little input data
- Generate from outlines
 - use building outline and height

My work

- extended the “generate from outlines” method to allow arbitrary polygonal meshes as the basis models
- implemented a flexible technique to generate geometry
 - system can be used for many different city styles
- integrated this system with a 3D editor

Arbitrary meshes



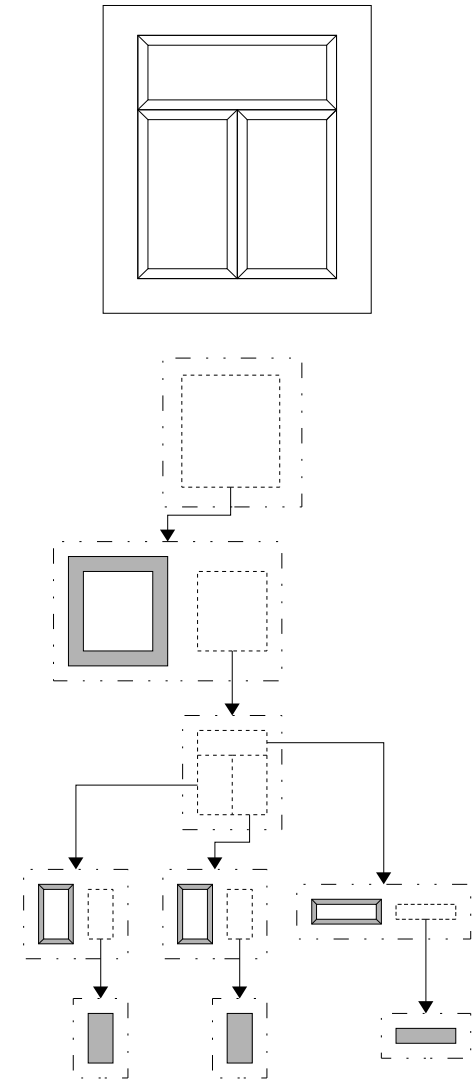
- triangles grouped into faces
- faces classified into “roof”, “wall” or “floor” based on their normal

Repetition

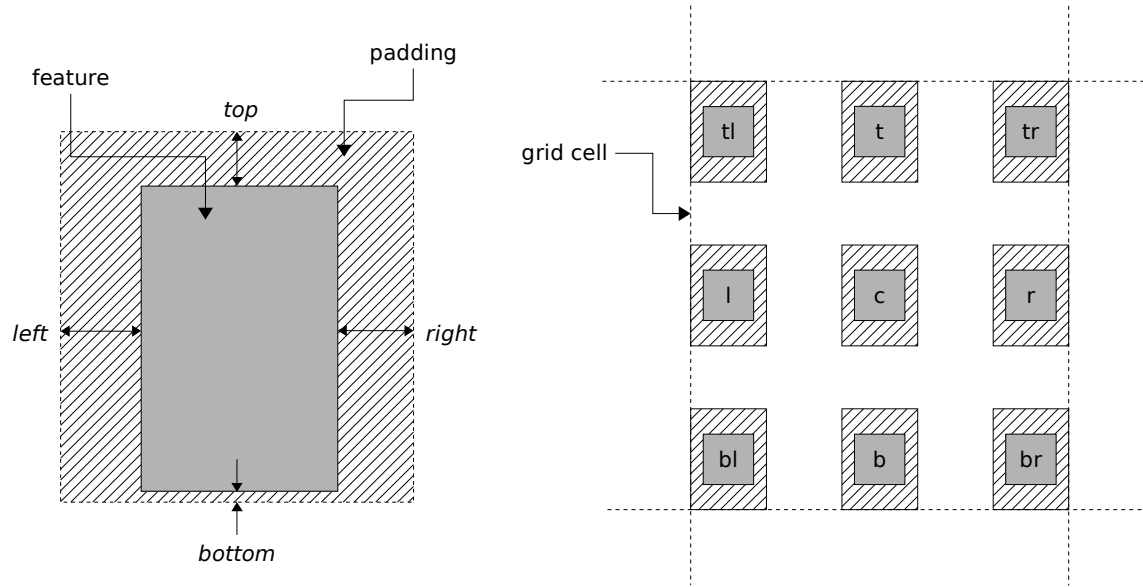
- Each building should look different, so we must avoid re-use wherever possible:
 - roof is generated per-building
 - features is also generated per-building
 - this has not been done before [to my knowledge]
 - not textures due to enormous load that would put on video memory
 - complex items of scenery need to be repeated rather than generated procedurally
 - in many cases it would be more expedient to create numerous versions of the same object for the various buildings than it would be to write a generation algorithm

Feature generation

- Features generated using a hierarchy of simple functions
- Core system knows nothing of the hierarchy- this just exists in the data set
- Features have several shared textures rather than a single custom one



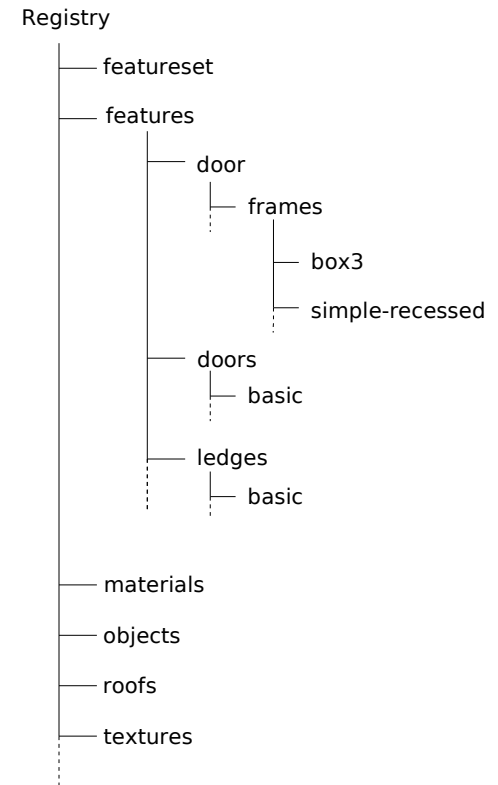
Feature Placement



- Features are arranged on walls in a grid
 - each feature has a justification and a padding value defined for it, and these are used to place features within grid cells
- An architectural style class [selected at random for each building] picks which feature should be placed in a given cell

The Registry

- The Registry is the database containing everything that will be used to generate buildings
- It is populated at run-time by a “dataset root module”
- It is arranged like a filesystem, with the path to an leaf node being its type (eg. “/features/door/frames”) and the leaf node being a container for the object itself

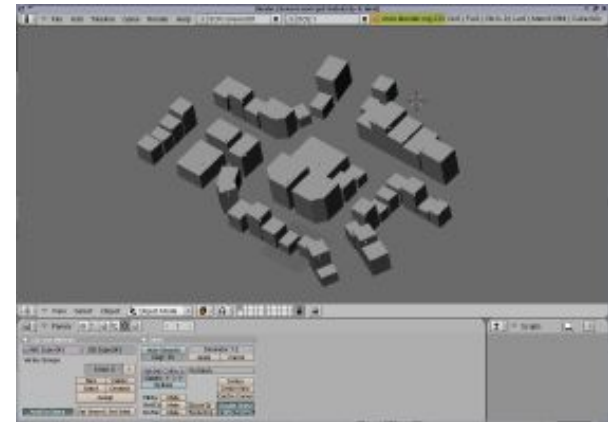


The Registry

- The data loaded at run-time also defines how the Registry is connected together
 - each object knows what the types (*ie.* paths) of its children are, and how to interface with them
 - this decouples knowledge of the data from the core system, making the system very flexible

Editor integration

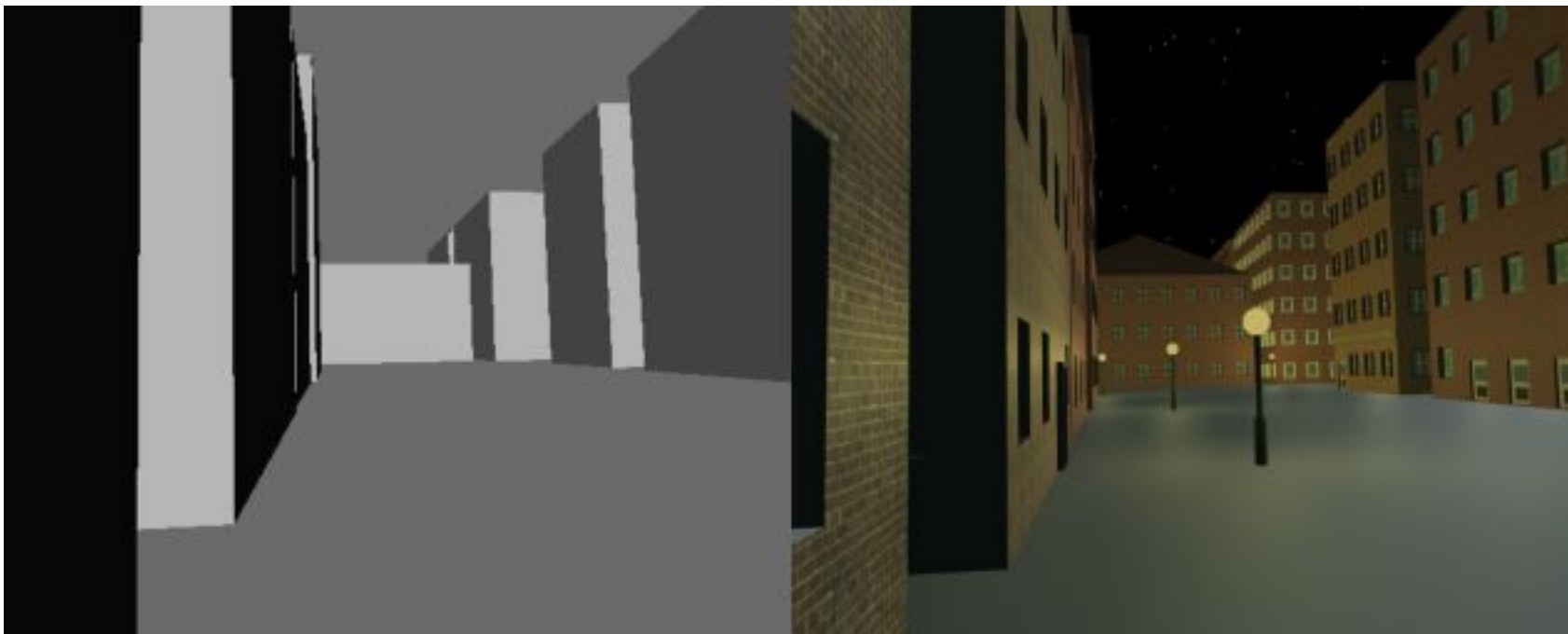
- The system is implemented as a standalone executable and as a plugin for Blender
- Editor integration improves the workflow of an artist using the system



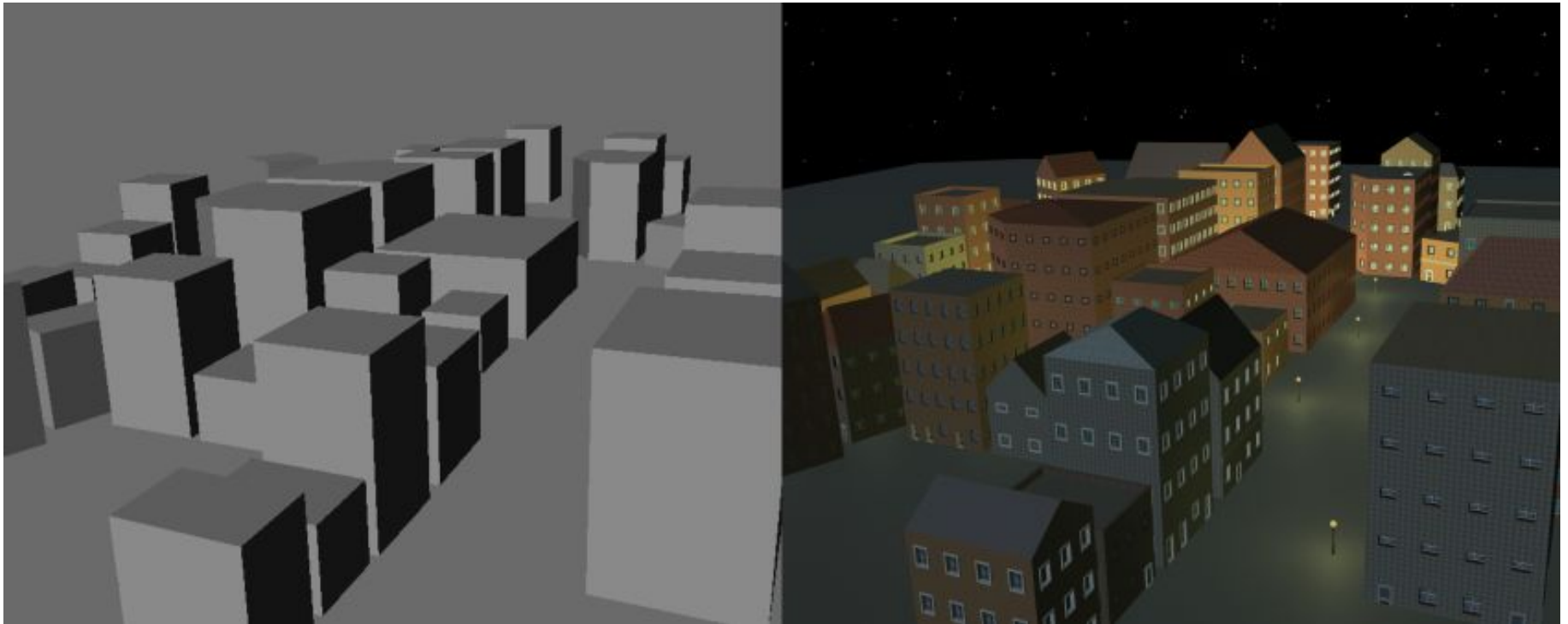
Results



Results



Results



Results



Conclusions

- The system was very successful: its output is of a good standard
- Main issue is the lack of control over the output: because of the heavy reliance on randomness, the Registry must be carefully designed to avoid stupid combinations

Further work

- The system could generate multiple levels-of-detail automatically
- The editor integration could be made far better:
 - reimplement the system as a two-phase process [parameter selection and geometry generation]
 - allow interactive access to the parameters to allow the user to “tweak” and see the results in realtime

Any questions?