Lessons in User Interface Design in the Procedural City Generation for Games Tool Ürban PAD

Lionel Barret [*] and Claudia Vance [†]
Gamr7 SAS
9 rue de la Monnaie
Lyon 69002, France
{lionel, claudia}@gamr7.com

G. Michael Youngblood [‡]
University of North Carolina at Charlotte
Computer Science, 9201 University City Blvd.
Charlotte, NC 28223-0001 USA
youngbld@uncc.edu

ABSTRACT

Procedural content generation design often involves configuring an array of abstract choices at each stage of the creation process. For new tools users, or for a non-specialist user, these choices are often complex and uncertain. A good interface that guides the user in their choices helps overcome this uncertainty and subsequent frustration with the process of generating procedural content. Since good user interface (UI) experiences facilitate adoption, procedural software tool developers constantly refine UI design in order to accommodate the engineering demands of procedural content generation architecture and translate them into a coherent user experience.

This paper discusses Gamr7's experience in creating, testing, and refining procedural user interfaces based on our experience with redesigning the user interface for our procedural content generation software, Ürban PAD. This case study will address user expectations, architecture, and execution flow considerations as well as solutions to some of Ürban PAD's specific user interface challenges.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: User-centered Design; K.8.0 [**Games**]; J. [**Computer Applications**]: Graphical Modeling Pipeline Tools

Keywords

Procedural City Generation, User Interface, Lessons Learned

1. INTRODUCTION

The games, simulation, and movie industries are in constant need for more and more art assets that form virtual worlds to support interaction, training, evaluation, virtual sets, and other uses. Consumer demand for richer and more expansive worlds coupled with the rapid advancement of computational ability to support this need has created a significant challenge to create content through crafted manpower, which is expensive and time intensive. Maintaining a manageable pool of art talent and sufficient tool pipelines simply will not scale with this demand.

A promising solution to the content creation problem is to automate parts of the content generation pipeline by capturing, abstracting, and parameterizing the process. This well-established process has been long used in computer graphics [1], which has been used for activities such as creating trees [6] (commercialized in SpeedTree [www.speedtree.com]), textures [5][1] (Allegorithmic's MapZone and Substance [www.allegorithmic.com] provide a commercial procedural texture generation tool), and even city environments [3, 2] (Procedural's CityEngine, [www.procedural.com], offers a commercial tool based on Müller's work—he is also CEO).

Building cities through traditional 3D modeling is difficult and time consuming; however, cities often follow basic patterns that could be replicated mathematically. Many games, simulations, and movies specifically need massive-scale human cities to be created rather quickly and inexpensively—making it an important and interesting problem domain. Instead of solely using shape grammars [3] that require detailed abstract configuration (programming in the shape grammar) and building façades [4] derived from existing building images, we have focused on working more closely with the artists to bring their uniquely crafted and stylistic ideas into the world of procedural content generation.

1.1 Ürban PAD

Our commercially available Ürban PAD (UP), which can be downloaded from www.gamr7.com on request, is a software tool dedicated to the creation of virtual cities. To produce such massive and complex scenes, UP uses procedural content generation to automate repetitive and time-consuming tasks in the creation of city buildings, roads, and other artifacts. The software is composed of many parts, each of them managing one aspect of scene creation with focus areas on: static mesh import, model creation, material definition, building templates, road layout, city generation, and scene exports.

Using the Ürban PAD tool begins with allowing the user

[*]Gamr7 Co-founder and Technical Director
[†]Gamr7 Marketing and Communications
[‡]Academic presentation and editing.

[1]There are hundreds of papers on procedural generation of textures.

to import 3D models and textures made with external tools. Procedural textures from Allegorithmic's Substance can also be directly included. All of these assets are collected in the project and made available to the builder components, which start with forming the building templates. Creating building templates is a process that defines the rules in which new geometry, imported geometry, and textures are allowably combined to create whole building structures. The next step in city creation is to define the roads, which can be either placed manually, established by a set of rules, or a combination starting with machine layout tuned manually. The structure of the roads defines sectors of the city in which to populate with the set of building templates previously defined or imported. The last step is to define the rules by which the city sectors are populated by buildings or other defined artifacts of the previous steps. The final result is a city generated by rules that can be recreated over a large number or permutations with a single press of a button on the interface.

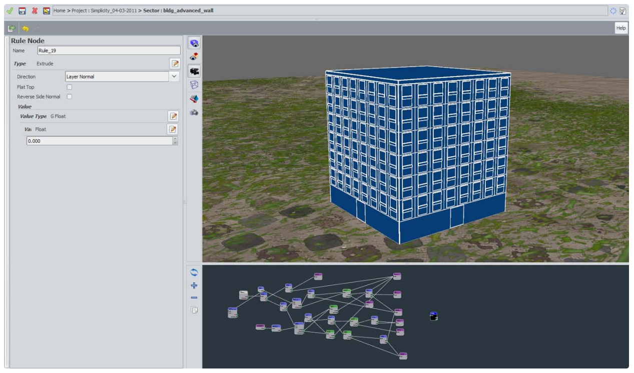
Producing a commercial procedural content generation tool that is integrated into the asset creation pipelines of media studios presents many challenges. One of the key challenges is adoption of the tool and integration into existing workflows. The user interface (UI) is the gateway to allowing users to use your tool and to enable the use of procedural content generation to meet their goals. This paper presents some specific development lessons learned over the past three years with regard to Ürban PAD.

We will focus on UP's Building Editor because it is one of the more unique facets of our work. This module is representative of UP's ability to provide control over the procedural content generation process and offers a good example of the UI design challenges facing UP, which call for learning the interface in one sitting without having to learn the whole framework. Users with modeling experience and basic knowledge of procedural generation expect these types of software tools to be familiar, intuitive, and fit organically into their asset pipelines.

2. PROCEDURAL BUILDING CREATION THROUGH TEMPLATES

The Ürban PAD Building Editor (BE) is focused on building model creation, similar to those created with 3D modeling software such as Autodesk's Maya or 3ds Max, by combining procedural geometry with asset placement. Buildings can be composed entirely of procedural geometry or have prebuilt model assets (e.g., custom windows, planters, or gargoyles) applied under arrangement rules. Specific model assets may be created in the BE or imported as mesh geometry with textures. A visual programming language is used to create and connect a series of transformation nodes that specify rules defining the configuration space of the building as shown in Figure 1. The resulting UP model, known as a template, can be used to obtain a variation of the specified building. Examples of variations include different levels of detail (LOD), different heights, different wall textures, and the same type of sub-components (like windows) with different appearances.

The Building Editor is one component of the UP suite, which contains content editors and a city generator. Editors are used to create urban content, such as buildings and roads, while cities are generated in a dedicated procedural generation engine. The overall UP workflow consists of the following steps, not all of which maybe used depending on the user's needs:


Home > Project : Simplicity_04-03-2011 > Sector : bldg_advanced_wall

Rule Node
Name: Rule_19
Type: Extrude
Direction: Layer Normal
Flat Top: ☐
Reverse Side Normal: ☐
Value
Value Type: G Float
Val: Float
0.000

Figure 1: The Ürban PAD Building Editor showing a rule definition panel on the left, and a procedurally generated building above the transformation graph—a form of visual programming that specifies the building configuration template.

1. Import externally created assets. A user's assets, generated with other software (Maya, 3ds Max, Blender) are imported to UP (textures, meshes). This step is not required as it is possible for a user to create a procedural building using only the procedural geometry creation features available in UP and the few basic textures furnished with the project package. Any imported assets will be packaged into the city project. The city project is the basic unit in UP and contains all files, which includes imported assets and UP-created templates associated with a city.

2. Creation of procedural geometry in the Building Editor through the use of rules. Rules transform geometry, which creates model layers. Rules, which are a form of pseudo-scripting, transform the layers on which they are applied, deforming them geometrically. Rules allow users to build variety into their templates as they work by entering randomized rule parameter values.

3. Decoration of procedural geometry in the Building Editor with the imported assets or procedurally generated sub-components. For example, this is the step where those fancy imported gargoyles are added to the building as defined by the rules set in this step.

4. The final model specification is a template. This template can generate specific instances of this model under the defined rules. All instances are present at once, encapsulated in the model structure and rules. This presence allows different variations to be generated as shown in Figure 2a and b.

5. The final model can be used in a city, generated by UP's City Editor component. The template can also be exported in Collada format as a standalone model as shown in Figure 2c.

In UP, a user does not create a specific model, but rather one abstract model/generator that, when used, will produce

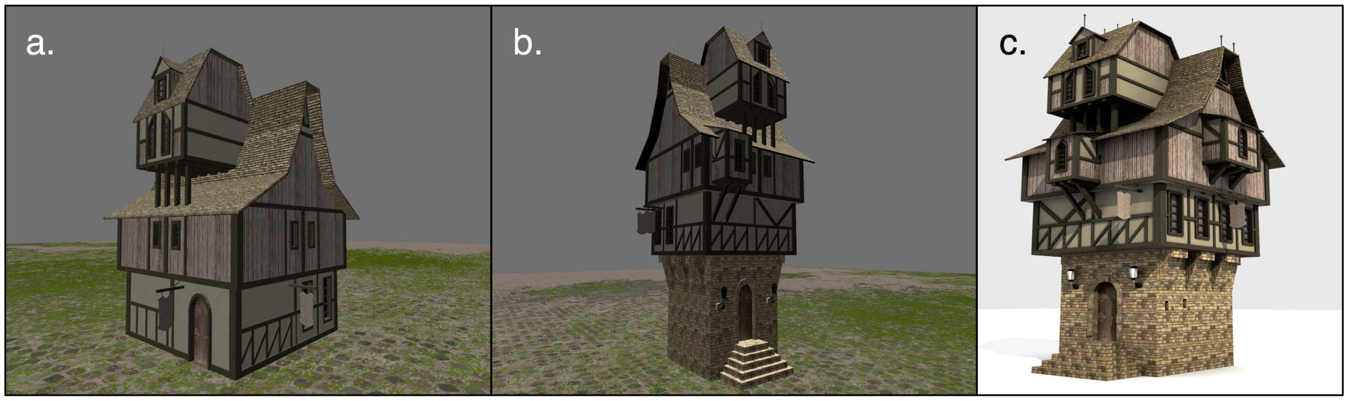


Figure 2: From the Ürban PAD Building Editor: a) one instance of a single model template, b) another instance with an optional part shown in configuration, and finally c) another instance exported in Collada format and rendered in Blender 3D.

specific instances. An example would be a model that has windows in one instance, but does not have them in another. In the template file, the possibility of both instances exists in the model simultaneously. However, the number of windows is instance-specific information, decided at generation time. Because an abstract model is generated and all instances are present in the template at the beginning of instantiation of a specific instance, there is no additional storage cost to scale since only the template and base components are stored. The core element of the template that stores the rules to perform transformation and assembly of the instances is the transformation graph.

2.1 User Interface Components

A primary component of Ürban PAD's current Building Editor UI is a transformation, or dataflow graph as shown in the lower right pane of Figure 3. The user connects transformation nodes together to create the building model template. These nodes contain rules that modify the preceding nodes. Each rule includes a set of parameters, which can be adjusted to vary aspects like geometry height, texture color, face selection, and so forth. Except output nodes (texturing, export nodes), each node can be thought of as a pure function.

Located above the node graph is a visualization module that displays instances generated by the model as shown in Figure 3. This module gives visual feedback about the current state of the graph and helps users evaluate the model/template. It is also possible to link one model/template to another via a rule, so another model/template can generate a subpart of the instance. One example would be linking a roof model, stored as a separate file, to a house model. In this case, the house model would be the parent model, and the attached roof model, the child model. The parameters of rules can be specified or adjusted when either a node or link is selected, which will populate the left panel of the UI as shown in Figure 3.

Model components can be arranged hierarchically, and some aspects of the child model can be overridden to provide a user with some flexibility in model editing or modification. For example, new contextual rules introduced in a recent beta version of UP allow the user to override the color of a child model with colors that match the parent model. To return to our roof example, a roof created with a selection of textures and linked to a house model will not necessarily include the match to the parent house texture at the moment it is linked as a child. Using contextual rules to override the color makes it possible to match roof model color to house model color.

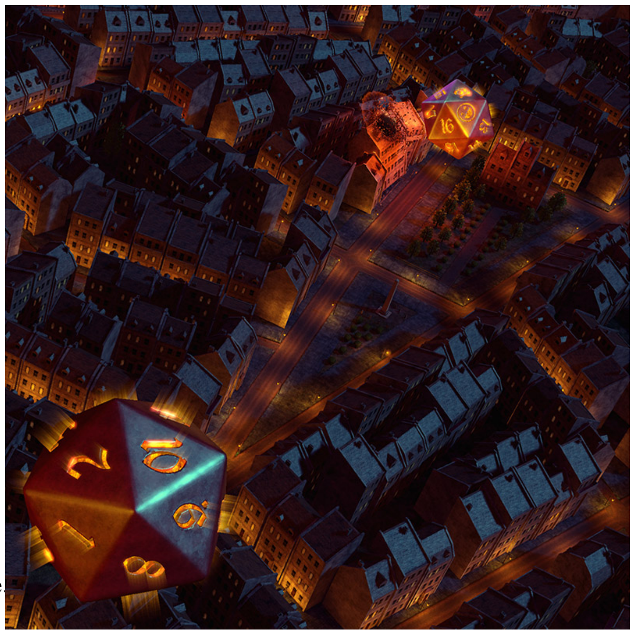
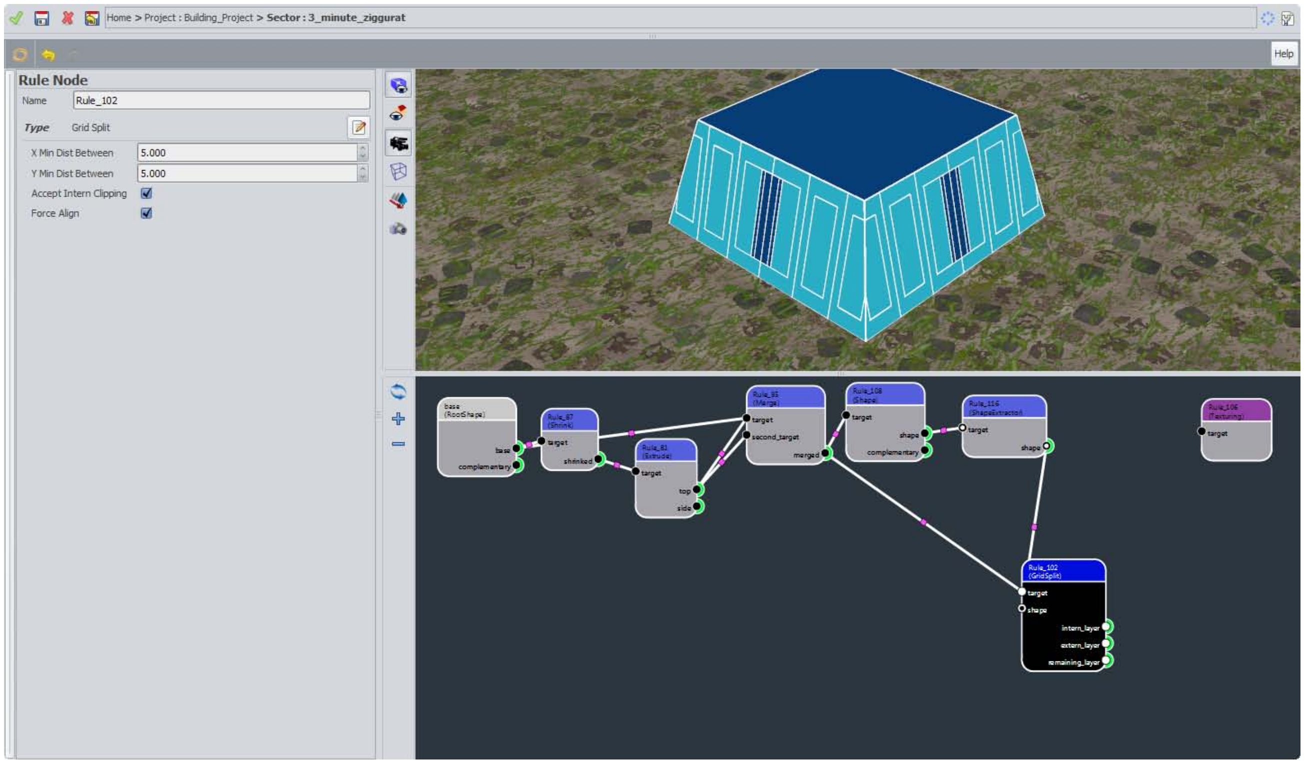


Figure 4: Ürban PAD was used by artist Histro Petrov for generating a city in his Unearthly Challenge entry rendered using Epic's Unreal Engine 3.

As we will learn, the development of the current transformation graph arose from user feedback. Faced with user content generation challenges and frustrated first-time users, we realized that the software needed refining. User feedback indicated that the UI was a viable solution for creating massive amounts of 3D content as demonstrated in Figure 4, but that lack of a user-friendly interface often frustrated these efforts.


Home > Project : Building_Project > Sector : 3_minute_ziggurat

Rule Node

Name: Rule_102

Type: Grid Split

X Min Dist Between: 5.000

Y Min Dist Between: 5.000

Accept Intern Clipping: ✓

Force Align: ✓

Figure 3: The Ürban PAD Building Editor showing a rule definition panel on the left, and a procedurally generated model above the transformation graph on the right.

3. USER DIVERSITY & FEEDBACK

In redesigning the user interface, we needed to consider different user populations. Interface construction therefore began with consideration of a user's mental model. Ürban PAD's user base includes beginner 3D content creators, programmers, and professionally trained 3D artists. Based on information culled from three years of informal interviews, we were able to further breakdown this latter category, which makes up the bulk of our user base, into nodal designers, which are technical artists who create a bridge between art (artists) and development (programmers).

Programers expected procedural content creation software to include scripting capabilities, while 3D artists were surprised to find that the software did not include a node-based graph. Both expectations reflect different populations' conceptions and expectations of what such software should look like and how its workflow should be structured. Not to be ignored, many of our users are beginners and have very little knowledge about 3D modeling in general. From our experience, the growth of the 3D modeler population comes from amateur and hobbyist circles. The success of Google's SketchUp modeler and Unity game engine seem to support this observation.

At one end of the spectrum, many programmers were interested in having a graphical REPL (Read-Evaluate-Print-Loop). Their mental models were based on this type of workflow, and some aspects of it were quite different from those designed in UP: sequential/synchronous (instead of asynchronous), debug-oriented, text-based, and so forth. On the other end of the spectrum, artists accustomed to immediate direct contact with the material and direct manual modification expected an interface based on a free-form approach on how to specify the model (e.g., something similar to Pixologic's ZBrush). In the middle of these two extremes was the nodal designer, less common but the most familiar with procedural content generation. Nodal designers are often technical artists responsible for finding and implementing technical solutions for art creation. They are interested in the extensibility of the software (adding new transformation nodes) and behave like expert artists when in front of a familiar UI (a dataflow graph).

Besides some professional views on the best interface to procedurally create buildings, most populations share some common expectations of 3D design software UI: undo/redo, copy/paste, text entry, or autosave. It is sometimes difficult to make these expectations compatible with the UP workflow.

The level of expertise is another variable that affects the perception—and a user's performance in using—of the UP interface. Artists who work heavily with 3ds Max, for example, will expect 3ds Max-style shortcuts.

Overall our user population draws from the following sources:

- Trial users, several hundred of whom actively provide comments. Trial users including technical artists in game development studios as well as non-specialist users who were interested in making 3D content.
- Commercial prospects gleaned from cold prospecting or that contacted us directly
- 3D artists in university/trade-school courses
- Industry partners from an ongoing collaboration in the

Rhone-Alpes region of France

- Gamr7's in-house 3D artist interns with knowledge of conventional tools like 3ds Max, Blender, Maya

When looking through our presentation notes and cold-call reports, we found that the following concerns applied consistently to all user populations. We decided that satisfying the following UI conditions was imperative for UP's success and adoption across a broad, diverse audience:

1. No requirement of programming knowledge or hard scripting. This was a particular concern of 3D artists, the bulk of our users, who may have had little or no exposure to computer programming or computational thinking.

2. Easy navigation (short paths) through the UI with instant feedback

3. Clear choices at each step of model construction and easy movement between them

4. Possibility of mastering UP framework through feedback and self-exploration

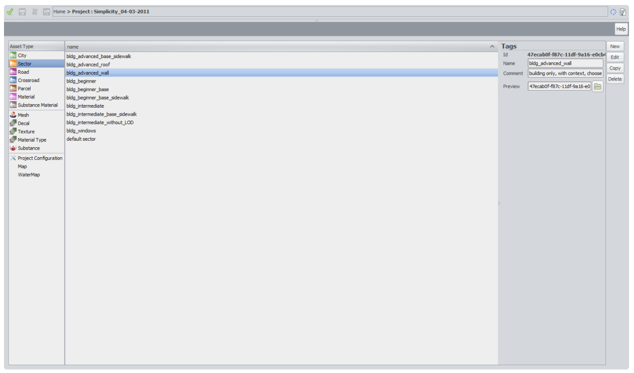
4. LESSONS LEARNED

Ürban PAD editing is a multi-task process. In order for users to have a coherent/satisfying experience with UP, the workflow must be designed so that users know where they are at any given time, and what they need to do next. Both considerations guided the reworking/redesign of the workflow based on user feedback.

4.1 Where am I?

Because UP contains multiple components, it is crucial for users to understand where they are in the workflow at all times. This concern was especially pressing in pre-3.0 versions of UP, as software components existed as standalone modules with no way of easily switching between them. Pre-3.0 versions of UP actually required users to manually open the correct editor, then choose to create a new template, or select an existing one to edit. Once a template was created or edited, the user saved it and closed the editor. If the model needed to be changed once the user was in another editor, the user had to save current work, exit the editor, and start up the original editor in order to change the template. The user also had to remember the template name, as there was no way for templates to be kept in the application's memory and be launched automatically. This process was time-consuming and broke the fluidity of the workflow, so we were obligated to redesign the workflow.

Saving models, manually closing and opening editors, and having to remember the name of the template to edit was addressed with the arrival of the action stack and database management in UP 3.0. The action stack allows instant processing of tasks depending on which asset is clicked from the main screen by keeping track of actions of the system and preserving actions in the editor save files. This is a LIFO stack that makes it easy to move between editors and keep track of actions. In UP 3.0, all assets are also tagged with a unique identifier and listed in the asset database, which is accessible from the main screen as shown in Figure 5. When an asset is clicked, the action stack immediately launches



Home > Project : Simplicity_04-03-2011

bldg_advanced_base_sidewalk
bldg_advanced_roof
bldg_advanced_wall
bldg_beginner
bldg_beginner_base
bldg_beginner_base_sidewalk
bldg_intermediate
bldg_intermediate_base_sidewalk
bldg_intermediate_without_LOD
bldg_windows
default sector

Id 47ecab0f-f87c-11df-9a16-e0cb
Name bldg_advanced_wall
Comment building only, with context, choose
Preview 47ecab0f-f87c-11df-9a16-e0

Figure 5: Ürban PAD's project interface, which auto-recognizes file types and loads the correct editor and provides uniform access to all assets.

the correct editor for the task and displays the hierarchy in a status bar.

This unifying act of bringing all of the editor and generator components of UP into a single interface that can recognize file types and maintains a database of all assets greatly improved user satisfaction and workflow.

Key Lesson: Provide easy workflow management and resource continuity/access in multi-stage tools.

4.2 What do I do next?

Solving the location problem and unifying the systems through a single interface opened up possibilities for us to focus on the overall workflow problem—helping the user figure out what to do next.

In the UP Building Editor, there are three possible types of next steps:

1. Start/complete a task: edit a sub-template or close the current one

2. Build/enhance the current template

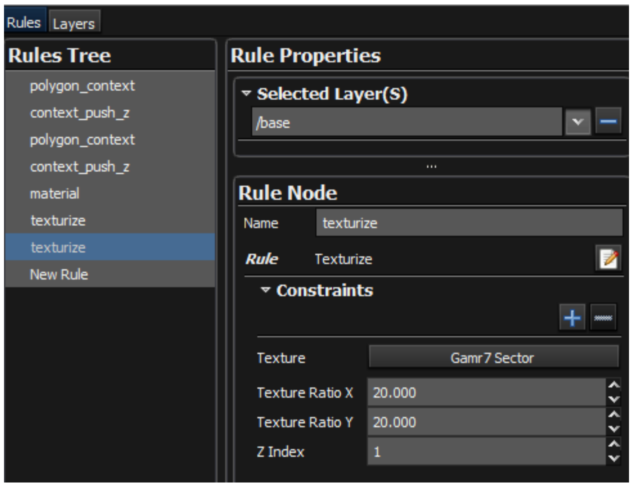
3. Export a template (a lateral task that neither enhances a template nor ends a work section)

4.2.1 Start/complete a task

Introducing the action stack also solved the editing workflow problem. Action stack processing makes it possible to start and finish tasks easily, and to move from editor to editor to change a model without having to close the original editor. A prime example of this function is the sector linking capability, which can be used to add building templates to a city sector in the UP Sector Editor. Instead of having to manually save templates and open the correct editors, users can access different templates and different editors without having to leave UP's Building Editor completely—just open the Sector Editor and add the current template then return to the Building Editor. The action stack will add and remove actions as necessary to accomodate the in-focus builder at any given moment.

4.2.2 Build/enhance the current template

To build or enhance the current template, users need to add rules to successive template layers. Each application of a rule transforms the layer(s) to which it is applied, creating additional layers of procedural geometry. In pre-3.0 versions of UP, rules were added in a stack-based system that relied on a highly abstract textual UI. In early versions of the pre-3.0 interface, users selected rules they wanted to use in the Building Editor from a dialog, then manually entered the rule parameter values and layer names in a separate field. This process was long, tedious, and frustrating for users accustomed to instant UI access and visual feedback.



polygon_context
context_push_z
polygon_context
context_push_z
material
texturize
texturize
New Rule

Selected Layer(S)	/base
Name	texturize
Rule	Texturize
Texture	Gamr7 Sector
Texture Ratio X	20.000
Texture Ratio Y	20.000
Z Index	1

Figure 6: Ürban PAD's 2.0 interface required correct sequential placement of rules in the rule hierarchy tree (left) and untyped/unrestricted value entry (right).

There was no way of visualizing connections between model elements, and proper model display in the very earliest versions depended on correct sequential placement of rules in the rule hierarchy tree to correctly carry out synchronous generation as shown in Figure 6. Model design in UP 2.5.x versions required selecting subsequent transformations in the correct order. Selecting a transformation that was not possible was likely to crash UP. There was no visual feedback on what chains of dependencies looked like, and users were required to tab back and forth through rules to access each piece of a dependency. If a user was familiar with UP and thought through their process execution correctly it worked well, but was admittedly rather unforgiving.

Early users indicated that the UI was both too abstract and not abstract enough: too abstract in that a text entry-based, hierarchical UI presented a very steep learning curve, as it presupposed familiarity with the range of rule choices available in UP and the correct rule parameters associated with them. Simplifying model construction would need to involve a non-linear workflow so that users could use a nodal graph-based interface that many of them has become accostumed to in other pipeline tools. This was the only viable solution for representing large amounts of visual information and being able to adjust them on an as-needed basis.

The current version of UP includes a graph-based, rather than a stack-based UI as shown in Figure 7. Transitioning to a graph gave users the possibility to use a non-linear workflow that better adapted to the needs of procedural content generation, with the ability to modify nodes, create branched dependencies, and edit model aspects that are not directly interdependent.

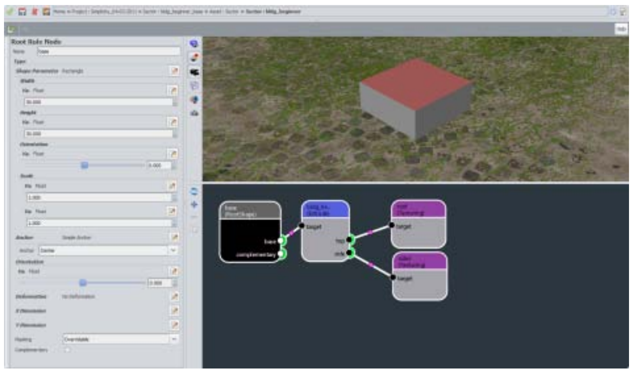


Figure 7: Ürban PAD's 3.0 interface with a nodal graph-based interface that supports non-linear editing.

A linear, synchronous workflow with stack-based rules kept the user bound to chains of dependencies that did not allow them to move back and forth to parts of the model that were not directly dependent on one another. The full UP2 Building Editor is shown in Figure 8. Changing a simple rule parameter involved tabbing through lines of rules, finding the rule to be changed, and adjusting the parameter. Progress in UP's current design stems from user feedback on earlier UP versions. Previous versions displayed available parameters as blank boxes into which numbers and/or text needed to be entered in order for the parameter to function correctly. Users had difficulty knowing which values to enter, or which parameter values were pertinent to the part of the model on which they were working.

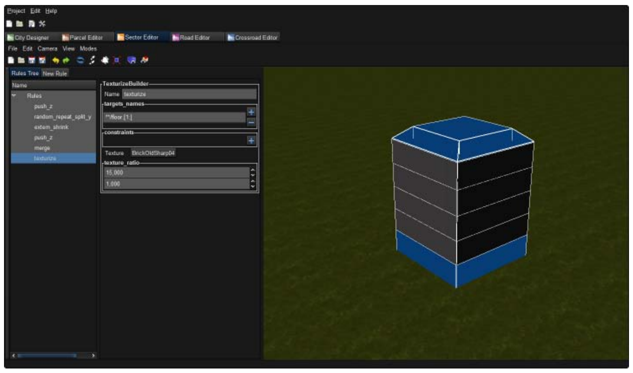


Figure 8: Ürban PAD's 2.0 Building Editor interface.

These ambiguities were partially addressed in UP3. A limited amount of information about which choice to make next is currently available via visual feedback. If a layer is affected by a transformation, the layer display mode shows the transformation. While this strategy does not provide explicit information about which subsequent steps may be logical in light of a preceding step, lack of visual feedback about a transformation may point to a problem in the usage of a rule node. This feature is currently being refined with the development of a dialog box alert system that will let users know when a rule node choice is invalid. Additionally, a variety of node rule information has been integrated into the rule parameters in the UI. Each time a node is selected, a rule parameter is displayed. Some parameters have been prefilled, indicating the places where a user will need to adjust the parameter in order to have visual feedback. However, some issues remain unresolved. Texture and material nodes are not obvious as end nodes, although for professional users they are logical. No changes are currently planned to address this issue, but we are monitoring user feedback.

4.2.3 *Export a template*

In contrast to start/complete or build/enhance models, exporting stands alone as a lateral task that neither starts nor ends a work session. In UP, users export models to Collada to check import into 3D rendering software like Maya or 3ds Max. After exporting, a user can come back to the template and keep editing. This task is integrated into the Building Editor, away from the two other task types.

Key Lesson: Assist users in complex workflows by providing easily understood contextual cues, feedback, and graphical over text displays.

4.3 Feedback and User Exploration

After introducing the action stack and an adapted workflow, we turned our attention to design improvements that would reduce the time needed to learn UP and enhance its fluidity for the user. This meant increasing UI feedback to help the user answer the question, "What did I do?" and master UP for quick, efficient content creation.

4.3.1 *Managing abstraction for feedback*

The current UP interface presents users with three kinds of abstractions: node-level, parameter-level, and model level. In older versions (< 3.0) of UP cited here, we consider that the node-level abstraction is a rule-level abstraction since the node graph did not exist in these versions. The first abstraction level, the node-level, refers to connections between applied transformations and their effects on a given model as shown in Figure 9. Node-level abstraction is the most immediate contact with the interface, as it is needed to understand a supplied model. The information contained in a node provides a user with specific, editable information about a node's properties such as height, radius, extrusion value, color, and so forth. Information display/representation at the node level is straightforward, but some visual encapsulation is missing, as it is not currently possible to hide, lock, or annotate groups of nodes.

The parameter level regroups all the parameters that supply specific information for a given transformation node as shown in Figure 10. The parameters are constrained and explicit. For example, parameters based on integers accept only entries of integers. They are the lowest/minimum abstraction level, and users can add new types of parameters as they are needed.

Model-level abstraction consists of connecting one model to another and overriding some parameters from the child model through a context function. In this case, the encapsulation is complete—as the implementation of the child model is hidden—but assumes that the user understands the whole abstraction gradient correctly. One aspect that confused users was the connection between model nodes. Models can be attached to other models via a linking rule, which places one model in another. This model-to-model linking has the same visual aspect as normal rule-to-rule linking, while being semantically and functionally different. For example, the blueprint view of a child model is not available in the parent model—only the final view is.

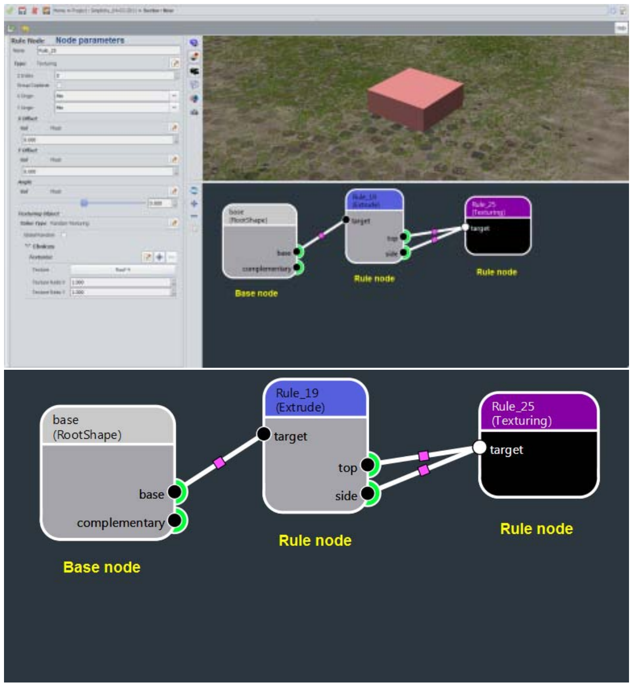
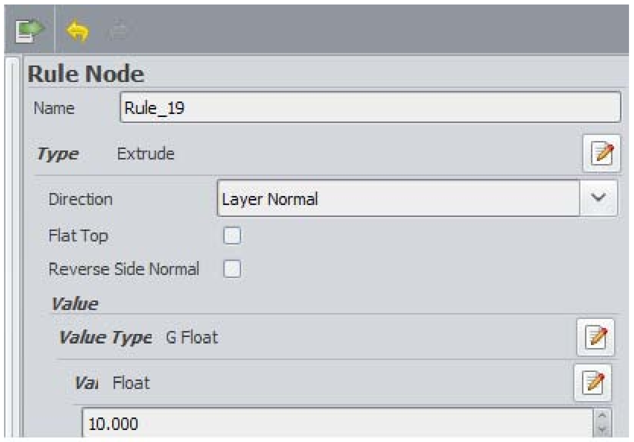


Figure 9: Node-level abstraction in Ürban PAD Building Editor.

4.3.2 *Visual Feedback and Annotation*

UP3's UI has been designed to obviate the need for handwritten annotations or design sheets by presenting node rule information visually as shown in Figure 11. The graph interface displays all node rules and the dependencies between them. Dependency paths are highlighted in white, making it possible for users to click on one part of the model and see the steps leading up to the realization of a given model component.



Rule Node

Name	Rule_19
Type	Extrude
Direction	Layer Normal
Flat Top	☐
Reverse Side Normal	☐

Value

Value Type G Float

Val Float

10.000

Figure 10: Parameter-level abstraction in Ürban PAD Building Editor.

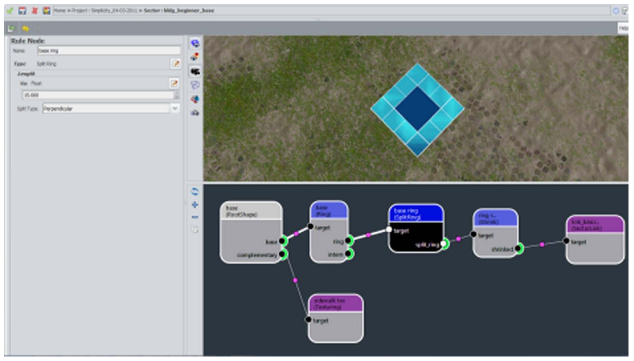


Figure 11: Ürban PAD Building Editor user interface with visual feedback and clear symbology.

Another area of improvement was the display of available rule choices. In previous UP versions, rule choices appeared in a simple dialog box with rules listed in alphabetical order. This system was unintuitive for most users, as it did not include any other kind of rule grouping (type of transformation or other classification). To make learning easier in UP3, we classified rules into families by transformation type. There are four rule families today: geometric transformation, decoration, resource generation, and filters (procedural selection). Instead of searching for a rule in a simple alphabetical list, users now learn four simple and easily searchable rule families. Rules appear in a dialog box when a node is created, and text describing the use of each rule when the rule is highlighted has been integrated into the dialog box. In the graph, nodes are color-coded according to rule family, making it easier for a user to search for a particular rule while editing.

Another significant problem this solved, predictable in hindsight but unanticipated, was a language problem. Gamr7 is a French company based in France, and the first users who worked closely with the program usually did not speak English as their native language. UP's interface is English-only, and these users were frustrated by having to learn alphabetically-grouped rules which had no other grouping that facilitated learning. In the very earliest versions of the Building Editor, there was no text describing the rules.

Hinting at the next step to take helps users understand better how to use UP. While previous versions displayed available parameters as blank boxes into which numbers and/or text needed to be entered in order for the parameter to function correctly, users had difficulty knowing which values to enter, or which parameter values were pertinent to the part of the model on which they were working. These ambiguities have been partially addressed in UP3. A limited amount of information about which choice to make next is currently available via visual feedback.

In beta tests of UP3, users asked for extended notation and additional function information like node and color grouping annotation. These seemingly small design improvements help users save time and navigate the interface more easily. The next version of UP will include a feature allowing users to add comments/annotations to nodes and groups of nodes. Recurring user feedback cited feeling lost in front of a new or large graph: the ability to read instructions and comments by the original graph designer was rated very highly as a helpful extension.

Key Lesson: Use color, hierarchical categorizations, and the appropriate level of abstraction to provide an organized and more intuitive workflow.

5. CONCLUSIONS

This paper introduced the Gamr7 Ürban PAD procedural city generation tool with a specific focus on the Building Editor. Procedural content generation configuration through the use of templates involving a graph-based specification of transformation rules using imported and/or procedurally generated geometry was presented along with the core user interface elements of the UP tool. Lessons learned from our first three years of tool iterations emphasize the importance of establishing a clear workflow with significant feedback to the user upon every action. Tool navigation should not be complicated, and artists and nodal designers (technical artists) highly prefer node-based graph editing as their preferred configuration paradigm. Additionally, we discussed the evolution of the UP Building Editor interface from user feedback

6. REFERENCES

[1] D. Ebert, K. Musgrave, D. Peachy, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach.* Morgan Kaufmann, 2003.

[2] G. Kelly and H. McCabe. Citygen: An Interactive System for Procedural City Generation. In *The 5th International Conference on Game Design and Technology Workshop*, 2007.

[3] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool. Procedural Modeling of Buildings. *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics*, 25(3):614–623, 2006.

[4] P. Müller, G. Zeng, P. Wonka, and L. V. Gool. Image-based Procedural Modeling of Facades. In *ACM SIGGRAPH / ACM Transactions on Graphics*, 2007.

[5] K. Perlin. Implementing Improved Perlin Noise. In R. Fernando, editor, *GPU Gems*, chapter 5. Addison Wesley Professional, 2004.

[6] P. Prusinkiewicz, J. Hanan, and A. Lindenmayer. *The Algorithmic Beauty of Plants.* New York, NY: Springer-Verlag, 1990.