Computers & Graphics 35 (2011) 352–363

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Semantic 3D Media and Content

A declarative approach to procedural modeling of virtual worlds

R.M. Smelik [a], T. Tutenel [b], K.J. de Kraker [a], R. Bidarra [b,*]

[a] Modelling, Simulation & Gaming Department, TNO, The Netherlands
[b] Computer Graphics Group, Delft University of Technology, The Netherlands

ARTICLE INFO

Article history:
Received 30 May 2010
Received in revised form
12 November 2010
Accepted 15 November 2010
Available online 23 November 2010

Keywords:
Virtual worlds
Declarative modeling
Semantic modeling
Consistency maintenance
Procedural methods
Procedural sketching

ABSTRACT

With the ever increasing costs of manual content creation for virtual worlds, the potential of creating it automatically becomes too attractive to ignore. However, for most designers, traditional procedural content generation methods are complex and unintuitive to use, hard to control, and generated results are not easily integrated into a complete and consistent virtual world.

We introduce a novel *declarative modeling approach* that enables designers to concentrate on stating *what* they want to create instead of on describing *how* they should model it. It aims at reducing the complexity of virtual world modeling by combining the strengths of semantics-based modeling with manual and procedural approaches. This article describes two of its main contributions to procedural modeling of virtual worlds: *interactive procedural sketching* and *virtual world consistency maintenance*. We discuss how these techniques, integrated in our modeling framework SketchaWorld, build up to enable designers to create a complete 3D virtual world in minutes. Procedural sketching provides a fast and more intuitive way to model virtual worlds, by letting designers interactively sketch their virtual world using high-level terrain features, which are then procedurally expanded using a variety of integrated procedural methods. Consistency maintenance guarantees that the semantics of all terrain features is preserved throughout the modeling process. In particular, it automatically solves conflicts possibly emerging from interactions between terrain features.

We believe that these contributions together represent a significant step towards providing more user control and flexibility in procedural modeling of virtual worlds. It can therefore be expected that by further reducing its complexity, virtual world modeling will become accessible to an increasingly broad group of users.

1. Introduction

3D virtual worlds have grown tremendously in size, detail and visual realism, their use being widespread far beyond entertainment games: they are found in games for training, movies, simulations, visualizations, online social spaces, etc. The downside of this, however, is that an ever increasing amount of content has to be created to fill up these worlds.

Current modeling systems offer designers total *control*, requiring the virtual world, with all its objects, to be modeled entirely by hand, on a low level of abstraction. With skill and dedication, designers can literally create any world exactly the way they want it, modeling every single aspect in detail. Eventually, the workload and repetitiveness of creating large worlds this way will become unbearable.

Virtual world models in manual modeling systems are also quite rigid: once completely constructed, they are hard to modify, e.g. major changes in the landscape may result in the designer effectively having to start from scratch. Furthermore, because manual modeling systems are complex and require extensive 3D modeling experience, the diversity of their user group is limited, typically excluding e.g. gaming enthusiasts creating new levels, and training instructors designing a tailored curriculum.

1.1. Procedural modeling

There is an urgent need for new modeling alternatives with both higher productivity and reduced complexity. Automatic procedural modeling seems an attractive alternative that promises such a productivity gain and a seemingly endless variation in content. It has been an active research topic for over thirty years, resulting in high-quality procedures for specific terrain features, such as landscapes [1,2], rivers [3–5], plant models [6] and vegetation distribution [7], road networks [8], urban environments [9], and building facades [10–12].

However, as concluded in our recent survey [13], traditional procedural methods are not directly a suitable alternative to manual modeling. They typically lack any user control other than

* Corresponding author. Tel.: +31 152784564; fax: +31 152787141.
E-mail address: r.bidarra@tudelft.nl (R. Bidarra).

0097-8493/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.11.011

a set of, often unintuitive, input parameters, which not always have a clear, predictable effect on the output, the generated content. To some extent, using traditional procedural methods in a modeling system comes down to *trial and error*. In addition, because the runtime of these algorithms often is far from interactive, this process becomes even more cumbersome. Furthermore, each procedure generates one specific type of content. Fitting all this content together into a consistent virtual world involves a large amount of manual effort. These causes explain why procedural modeling is hardly used so far in mainstream content design. A hybrid solution combining the strengths of procedural methods with manual control would be ideal.

1.2. Extending traditional procedural methods

In recent years, the research direction in procedural modeling is shifting to user controllable and interactive procedures, thereby successfully addressing some of the downsides of traditional methods. Here we survey some noteworthy examples.

The lack of user control in procedural generation of elevation maps was the first issue addressed by several researchers. Their methods vary in interactivity and level of control, from coarse to fine-grained. Some of the proposed extensions provide a way to constrain the generation process in a non-interactive manner by new forms of user input. Stachniak and Stuerzlinger [14] propose a method that integrates constraints expressed as mask images. It employs a search algorithm that finds an acceptable set of deformation operations to apply to a random terrain in order to obtain a terrain that conforms to these constraints. Zhou et al. [15] describe a technique that generates terrain based on an example input height-map and a user line drawing that defines the occurrence of large-scale curved line features, such as mountain ridges. Features are extracted from the example height-map, matched to these curves and seamed in the resulting height-map. Saunders [16] proposes a method that synthesizes a height-map based on Digital Elevation Models (DEM) of real-world terrain. A user draws a 2D map of polygonal regions, each of which is marked to have a certain elevation profile. A height-map is instantiated using a genetic algorithm, which selects DEM data that matches the requested elevation profile. Kamal and Uddin [17] present a constrained mid-point displacement algorithm that creates a single mountain according to such properties as elevation and base spread. Belhadj [18] introduces a more general method where a set of known elevation values constrain the mid-point displacement process. Doran and Parberry [19] propose a different constraint-based approach using agents, each creating a specific landform (e.g. coastline, beach, mountain).

With the evolution of the GPU as a device for general purpose parallel processing, interactive user control in elevation map generation has become feasible. Schneider et al. [20] introduce a setup in which the user interactively edits the terrain by painting grayscale images, which are used as the base functions of their noise generator. Using an efficient GPU-based hydraulic erosion algorithm, Stava et al. [21] propose an interactive way for users to modify terrain using several types of hydraulic erosion. To provide users with more control over the exact appearance of mountain ranges, Gain et al. [22] introduce a sketch-based height-map generation method in which users sketch the silhouette and bounds of a mountain in a 3D interface, and the generator creates a matching mountain using noise propagation. Even more fine-grained control over the shape of mountains is provided by the interactive procedural brushing system introduced by de Carpentier and Bidarra [23]. These GPU-based procedural brushes allow users to interactively sculpt a terrain in 3D using several types of noise.

Extensions of traditional procedural methods are also proposed for other terrain features, for instance for interactively defining road networks used in city models. Chen et al. [24] propose interactive modeling of road networks using tensor fields that can create common road patterns (grid, radial, along a boundary) and blend these in a plausible way. McCrae and Singh [25] present a method for converting strokes to 3D roads that are automatically fit in the terrain. Their system also creates junctions and viaducts for crossing roads. An A* based road generation method proposed by Galin et al. [26] uses a elaborate cost function to encode the influence of terrain slope, water bodies and vegetation on the trajectory of the road.

As districts, blocks and parcels are defined by the city's road network, Kelly and McCabe [27] propose an interactive method to generate secondary roads and house blocks based on the primary roads the user manipulates. A similar system by de Villiers and Naicker [28] lets users create a road network and city blocks using sketch strokes, and interprets a set of sketch gestures that modify the properties of the city blocks (e.g. population size, function). Weber et al. [29] present an interactive simulation system for cities growing over time, by expanding streets in the city's road network. A dynamic system that connects geometrical with behavioral modeling is proposed by Vanegas et al. [30]. Here, users paint behavioral variables like employment density, which automatically leads to changes in the population distribution and, thereby, the city geometry. Shape grammars are often employed for automatic creation of building facades (see e.g. [11]). However, defining a suitable shape grammar is complex and requires much experience. Addressing this, Lipp et al. [31] propose a shape grammar editing system, in which the effects of new rules are interactively visualized.

Although these extensions have clearly contributed to procedural modeling research, their downside is that they are primarily designed to generate one specific aspect of virtual worlds. Seldom has attention been given to the integration of separately generated terrain features into a complete virtual world. Early work by Amburn et al. [32] already formulated the problem of fitting roads with terrain: on a coarse level, the road follows the elevation profile of the terrain and on a fine level, the terrain must be modified to match locally with the road embankment profile. This specific integration problem was recently addressed by Bruneton and Neyret [33], who propose a shader-based system for real-time integration of Geographic Information Systems (GIS) vector features, such as road and rivers, into a DEM. They create a road profile texture based on footprint geometry, and integrate the profile by blending this texture with a height-map texture. The discussed work by Galin et al. [26] extents this by also removing any vegetation along the road. Although this research successfully addresses an important integration issue, managing the consistency of all terrain features as well as their relationships and dependencies remains an open problem.

A more generic approach to the problem of integrating terrain features and maintaining their relationships is to enrich their description with additional *semantic* information. Semantics-based modeling has been successfully applied to several fields, as surveyed in [34], including CAD/CAM [35], smart object behavior [36], and automatic layouts of interior scenes [37,38]. However, the role of semantics in procedural modeling has, until now, been limited, which hinders the systematic integration of all the different procedural methods.

Commercial procedural modeling tools typically also focus on generating one specific feature, such as height-maps (e.g. L3DT [39], and many others), vegetation (e.g. XFrog [40]) or city models (e.g. CityEngine [41]). A noteworthy exception is CityScape [42], which allows for a hybrid of manual and procedural modeling, although it provides a somewhat limited and narrowly focused set of procedural operations.

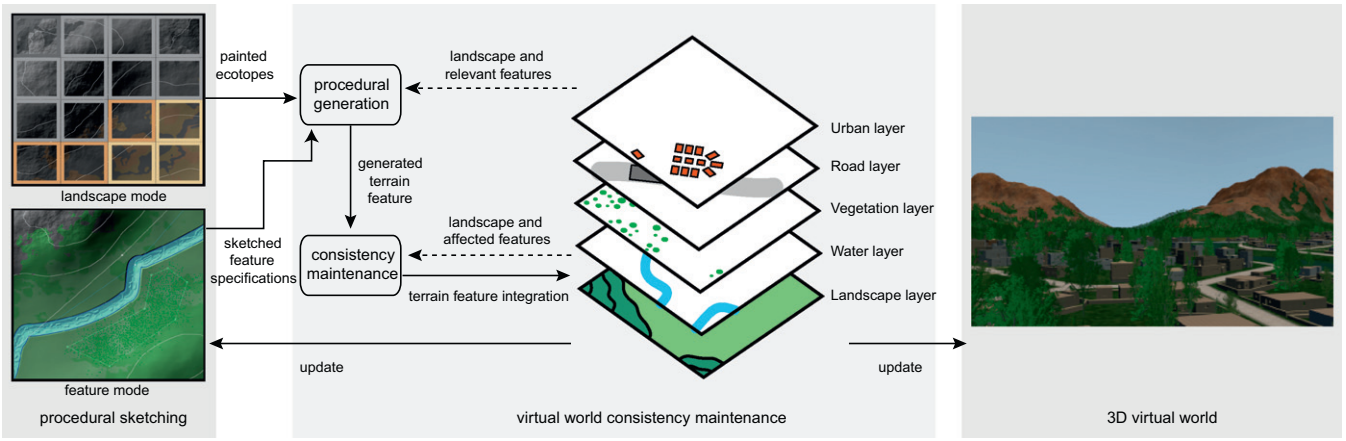

Fig. 1. An overview of the workflow of the declarative modeling approach: using procedural sketching (Section 2), designers interactively create the virtual world, of which each feature is automatically generated, integrated and maintained within the virtual world model (Section 3). From this semantic model of the virtual world, other representations are derived, such as a 3D geometric model.

1.3. Declarative modeling of virtual worlds

As stated above, procedural modeling research is no longer solely focused on the generation of individual models of high quality. Researchers realize that more intuitive input, improved user control, and automatic integration of results are instrumental to the acceptance of procedural methods in mainstream virtual world development. However, to date, no research method or commercial tool provides such an integrated and flexible solution that allows designers to procedurally model a complete virtual world that matches their intent.

Our *declarative modeling approach* aims at contributing to these open issues, by combining the strengths of manual and procedural modeling, and providing a more productive and less complex workflow to model virtual worlds. Basically, this approach lets designers concentrate on *what* they want to create instead of on *how* they should model it. Designers state their intent using simple, high-level constructs. The declarative approach builds upon established research results on parameterized procedural generation, constraint solving and semantic modeling, in order to automatically translate these statements into a matching 3D virtual world. The consistency of this world is automatically maintained using a *semantically rich* model of all its features and their relations, analogous to the automatic maintenance of interior scenes based on object semantics [37].

This article discusses the two main contributions of our declarative modeling approach:

1. an intuitive and accessible user interaction method called *procedural sketching* (Fig. 1 left hand) and
2. automatic *virtual world consistency maintenance* through generic methods for resolving interactions between terrain features and the landscape (Fig. 1 middle).

We are developing a prototype modeling framework, called SketchaWorld, that demonstrates the feasibility of this declarative approach (see Fig. 1). Its goals are:

1. to increase designers' productivity, while still allowing them to work in an iterative manner and exercise control over the generated results;
2. to provide an intuitive way for people without special modeling expertise to create virtual worlds that meet their requirements; and
3. to facilitate the application of results from research in procedural methods in an integrated modeling approach.

The remainder of this article is structured as follows. Section 2 presents interactive procedural sketching. Automatic virtual world consistency maintenance is explained in Section 3. The implementation and results of the prototype framework are shown in Section 4, including an example modeling session. Section 5 discusses the advantages and current limitations of our approach. Section 6 summarizes ongoing and future work.

2. Interactive procedural sketching

User input and control is provided by *procedural sketching* (Fig. 1 left hand). With easy to use editing tools, designers create a 2D digital sketch: a rough layout map of the virtual world. Procedural sketching provides two interaction modes:

Landscape mode: Designers paint a top view of the landscape by coloring a grid with ecotopes (an area of homogeneous terrain and features). These ecotopes encompass both elevation information (elevation ranges, terrain roughness) and soil material information (sand, grass, rock, etc.). The grid size is adjustable and the brushes used are similar to typical brushes found in image editing software, including draw, fill, lasso, magic wand and transition pattern brushes (e.g. from ocean to shore).

Feature mode: Designers specify features like forests, lakes, rivers, roads, and cities on the landscape using vector lines and polygon tools. This resembles vector drawing software: placing and modifying lines and polygons is done by manipulating control points.

Each sketched element is procedurally expanded to a corresponding terrain feature (*procedural generation* box in Fig. 1). To directly see the effect of an edit action on the virtual world model (e.g. drawing ecotopes, rerouting the path or modifying the shape of a feature, removing a feature), users sketch on a 2D top view of the generated virtual world. This view updates immediately as new results are generated. Depending on the interaction mode, an overlay is displayed representing relevant elements of the user sketch. Fig. 2 shows the user interface for procedural sketching in feature mode. By keeping the user interface and interaction modes simple and clear, procedural sketching is more accessible for people without special modeling expertise; see also [43].

A *short feedback loop* between a designer's edit action and the visualization of the generated results is essential to allow designers to model virtual worlds iteratively. This requires each edit action to be executed separately and the results of an action to be displayed immediately. Such an interactive setup allows designers to quickly

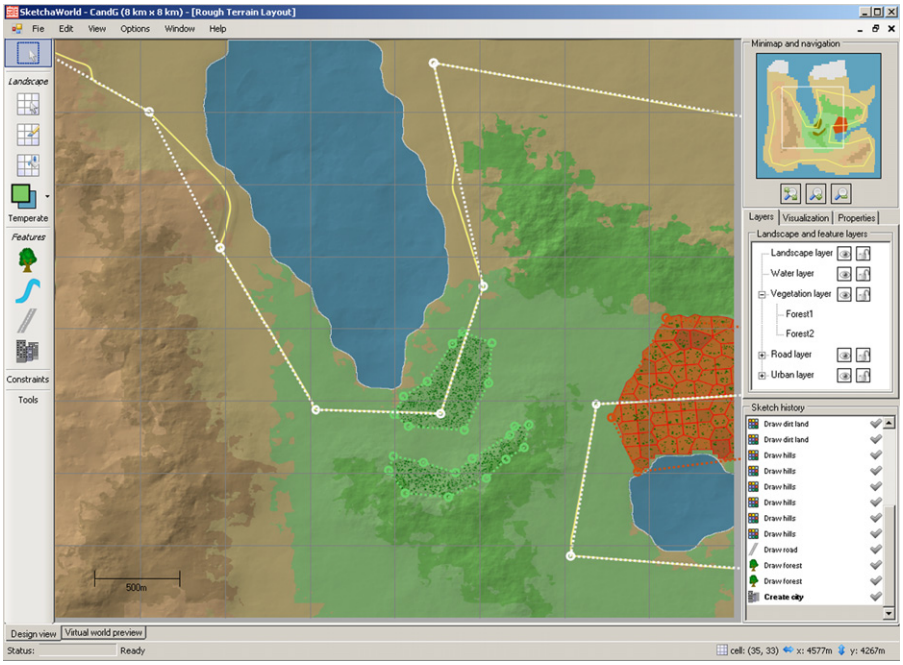

Fig. 2. User interface for procedural sketching (feature mode), also showing editing tools (left hand) and navigation, layers, and edit history (right hand).

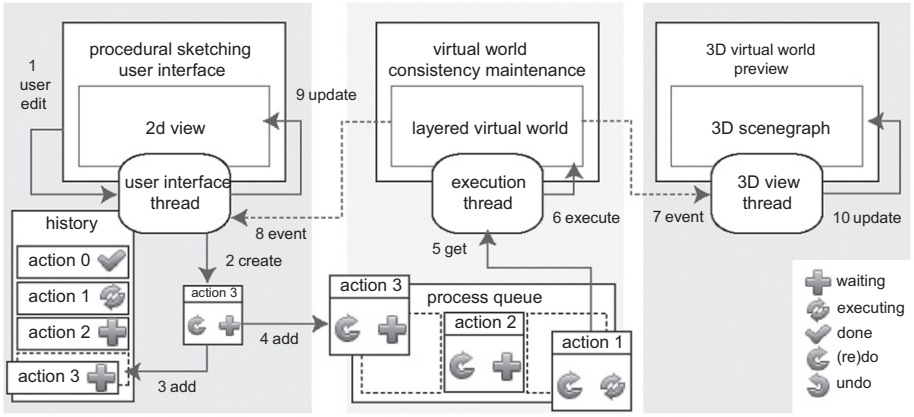

Fig. 3. Diagram showing the execution flow of interactive procedural sketching. In this situation, a designer has just edited (1) the landscape or a certain terrain feature. For this, the *user interface thread* creates an edit action (2) with id number 3 and appends it to the history (3). Furthermore, a copy of the action is enqueued (4) in the *process queue*. Simultaneously, the *execution thread* polls the queue (5) and obtains action 1, which is executed (6), modifying the virtual world. These modifications trigger events indicating changes (7, 8) that are received by the user interface thread, which updates the *2D view* (9) and the *3D view* thread, which modifies (10) and renders the 3D scenegraph.

see the effect of their edit operations and work towards the desired end result.

Several challenges have to be overcome in order to provide designers with this interactive and iterative workflow. The solutions for this are detailed in [44]. Although improvements in hardware and new approaches such as GPU computing significantly alleviate the execution time of procedural methods, operations affecting large regions or requiring complex algorithms (e.g. city generation) may still execute at non-interactive rates. Therefore an asynchronous setup was implemented, explained in Fig. 3. It separates the *user interaction* and the actual *execution* of edit actions. This allows designers to continuously work on the virtual world without being hindered by long procedural operations.

The ability to undo and redo any modeling action is one of the main requirements for an iterative modeling workflow. For this, the familiar edit history is provided. Because of memory constraints inherent to modeling large virtual worlds, it is far more efficient to implement undo and redo by (partial) regeneration, instead of storing all intermediate modeling states. At the expense of some additional computation time, designers are provided with unlimited undo and redo facilities. Furthermore, because designers expect regenerated results to be exactly the same as before when redoing an action, we need to carefully manage the state of the random number generator used in procedural methods, as explained in [44].

3. Virtual world consistency maintenance

Using procedural sketching, designers declare the properties of the landscape and specify its terrain features. In the declarative approach introduced in Section 1.3, these features are not only generated according to the user specifications, but they are also properly integrated in order to form a consistent and lifelike virtual world. Each terrain feature introduced in a virtual world affects in some way the existing landscape and nearby features, and vice versa, e.g. the feature fits itself to local constraints, it affects the structure of a nearby

feature, it modifies the elevation profile, or it forms some sort of connection with a compatible feature. One can imagine the amount of tedious manual modeling work if the responsibility of solving these interactions and keeping the virtual world consistent would be left to the designer. Because the semantics of terrain features and their relations are encapsulated in the layered virtual world model, we are able to maintain the consistency in an automated manner. This section summarizes the generic resolution methods with which the consistency of the virtual world model is maintained.

3.1. Virtual world semantic model

The virtual world's semantic model is a layered data structure (Fig. 1 middle). All terrain features are grouped in logical layers of the virtual world model, which are inspired from Geographic Information Systems. This ordering has been chosen because of the semantic similarity and relations of the terrain features within each layer. We distinguish five specific layers, stacked as follows:

1. Urban layer: e.g. cities, districts, parcels, buildings.
2. Road layer: e.g. highways, local roads and streets, bridges.
3. Vegetation layer: e.g. natural forests, planted vegetation.
4. Water layer: e.g. rivers, canals, lakes, oceans.
5. Landscape layer: elevation profile and soil material.

Each terrain feature is defined at several *levels of abstraction*, giving structure to its layout (e.g. city topology) and its objects (e.g. buildings). Procedural generation of each terrain feature can therefore be described as a top down process, starting from a coarse user specification, and refining this specification from abstract structures to concrete objects in several refinement steps, resulting in the complete feature representation. Moreover, the objects integrating a terrain feature do not only have a geometric description (e.g. polygon, spline, 3D shape), but also a *semantic* description, which includes their relevant attributes, as well as the logical connections and geometric and functional constraints involving them [37]. The number of levels of abstraction varies according to the complexity of the feature. At least, the following three levels can be discerned per feature type:

1. Specification level: user-sketched coarse outline and input parameters (e.g. a forest specification).
2. Structural level: the layout of the feature and the area it encompasses (e.g. the contour of the forest).
3. Object level: all individual semantic objects making up the feature that will result in concrete, geometric objects (e.g. the set of individual trees).

On the structural level, features only have a semantic representation; this structure helps to layout their individual objects, and to preserve their logical and functional structure (e.g. which districts make up a city center). At the object level, further structuring is provided by connections (e.g. street connectivity) and constraints (e.g. minimum distance between certain objects) between semantic objects.

3.2. Consistency maintenance basic notions

We first introduce several preliminary concepts and notions before discussing the actual resolution methods. In each refinement step in a terrain feature's generation process, we can discern two phases (shown in Fig. 1 middle):

1. Starting from the specification level, the next level of abstraction of the feature is procedurally generated.
2. This level of the feature is fit with its surroundings.

The generation procedure is steered by the provided specification and influenced by relevant nearby terrain features. In order to fit with its surroundings and place its objects, a feature often requires a certain area of the landscape to be clear of any other features. Moreover, the local elevation profile might be unsuitable for a feature to attach its objects; therefore, it might require the landscape to adhere to a profile constraint (e.g. regarding slope, regularity or flatness). For these requirements, features are provided with two types of requests:

Claim: A feature can make a claim for an area of terrain in order to use it exclusively. A *claim* is a request by feature f_x to reserve an area of terrain a (defined as a complex 2D polygon) for exclusive use at the given level of abstraction l. A claim can be either granted or rejected. A feature can claim an area of terrain on either the structural or the object level of abstraction. This allows a feature to disregard irrelevant, small objects when laying out its structure, and to consider only concrete, existing objects when placing its integrating objects.

Modification: A feature can request a local modification of the landscape (elevation profile or soil material) in order to properly fit with its requirements (see Section 3.3).

A *feature interaction* is said to occur when two claims are made for the same area of terrain. More precisely, a *feature interaction* is an overlap area a between the area claimed by feature f_x and the area already granted to another feature f_y, due to a claim at the same level of abstraction l. Because area ownership is by definition exclusive, feature interactions always have to be solved; we do this in one of two possible ways:

1. An interaction is solved in a cooperative way when it is possible to introduce a connecting structure. A *connection* formed at a level of abstraction l links the feature f_{lose}, for which the claim is rejected, to feature f_{win} over the disputed area a. The claim of f_{win} on this area a is or remains granted. Connections made on the structural level are abstract (e.g. a highway is linked to a city), and results in concrete connection objects (e.g. a road junction object), once made on the object level. Examples of connection objects include bridges, tunnels, road junctions, estuaries, etc. Of course, connections cannot be defined for every possible pair of feature types, as there might not be a sensible real world equivalent (e.g. between a lake and a forest). Furthermore, for some pairs of terrain feature types, several alternative connection types may be defined.
2. An interaction is solved in a conflictive way when one of the features, f_{win}, prevails and the other, f_{lose}, can no longer make any use of the disputed area a. If the conflict occurs at the structural level of abstraction, this entails that the feature layout of f_{lose} cannot include this area a and as such has to be restructured. At the object level, this means that the objects of f_{lose} cannot be placed within this particular area.

The decision mechanism for resolving interactions is steered by *priorities*. A priority is a ranking score or weight of a feature at the structural or object level. Priorities are divided in three categories:

- priority_{claim}(feature f_x, level l) is the priority of feature f_x for claiming an area of terrain at the level of abstraction l;
- $\text{priority}_{connect}$(feature f_x, feature f_y, area a, level l) is the priority of feature f_x for forming a connection with feature f_y at area a and at abstraction level l; and
- $\text{priority}_{conflict}$(feature f_{lose}, area a, level l) indicates the *threshold* at which feature f_{lose} rejects any connection.

For each feature type and for each level, a single user-defined value serves as the default priority value for every feature instance. However, for $\text{priority}_{connect}$ and $\text{priority}_{conflict}$, features can define *context-dependent* functions that compute the actual priority value,

for instance a cost function for road connections. Furthermore, each specific feature instance may override any of the priority values of its type.

3.3. Landscape interaction

Before discussing how multiple features interact with each other, we analyze the basic interaction between the landscape and a single terrain feature. The landscape plays a special role in our virtual world model in the sense that it forms its omni-present basis on which all other terrain features attach. Therefore, the landscape does not compete with terrain features in the form of claims, and no priorities need to be defined for the landscape. A local change to the landscape affects all terrain features in that area, each to an extent that is defined by its particular semantics. This notion is captured in the landscape interaction resolution method, summarized in Algorithm 1.

Algorithm 1. Landscape interaction resolution method.

```
// solve all interactions between landscape and terrain features
// a - area of the landscape that is changed
SolveLandscapeInteractions(area a):
    // find all affected terrain features
    F = {f | f ∈ features, a ∩ f.area ≠ ∅}
    sort F according to highest priority_claim(f, level_structural)
    // let each affected feature handle the landscape interaction
    for all feature f in F do
      f.restructure(a, level_structural)
    end for
```

As follows from this algorithm, the outcome of the landscape interacting with any feature f present in area a is always that f needs to adapt its structure. However, features decide to what extent to restructure, if at all, which typically depends on the scope of the changes to the landscape. Drastic changes in the elevation profile will probably cause features like roads or cities to strongly revise their structure, whereas changes in soil material will probably affect vegetation the most. Depending on the feature's procedure implementation, restructuring could entail that part of the former structure and objects of the feature are removed from the respective terrain layer(s), and that corresponding area claims are abandoned, thus allowing other features to reclaim these areas. We handle interactions ordered by priority_{claim}, as its value provides a good heuristic of the impact a feature will have on other features.

Although features adjust their structure to the landscape profile, to be able to attach to the landscape, features can have specific requirements for the local elevation profile. For instance, a building could require an area of flat terrain. To fulfill these requirements, features can act on the landscape through *modification* operations, which they can issue after each refinement phase. A *modification* is a request by a feature f_x for a certain area a of the landscape to be constrained to the given elevation or soil material profile p. This profile is to be integrated in the landscape according to a blending fall-of mask m. This mask allows for smooth transition zones. Such a modification request is combined with the overlapping requests of nearby features; claimed areas cannot be affected by other feature's modify requests.

3.4. Feature interaction resolution

Terrain features compete with each other to claim areas of the landscape for their own use. A generic interaction resolution method has been devised to handle potentially conflicting claims between terrain features. This resolution method is outlined in Algorithm 2, and works as follows. A new terrain feature f_x can make a claim for a certain area of terrain a_x, on either the structural or the object level l. The set of interacting features is found, and is sorted according to priority_{claim}, where features with higher priorities are resolved first, as these features will most likely have the highest impact on the new claim.

Algorithm 2. Feature interaction resolution method.

```
// solve all interactions between a feature and existing features
// f_x-feature which has made a new claim
// a_x-area of terrain claimed by f_x
// l-level of abstraction
SolveFeatureInteractions(feature f_x, area a_x, level l):
    // find all interacting features with granted claims
    F = {f_y | f_y ∈ features, a_x ∩ f_y. area ≠ ∅}
    sort F according to highest priority_claim(f_y, l)
    // handle all feature interactions
    for all feature f_y in F do
        if priority_claim(f_x, l) > priority_claim(f_y, l) then
            SolveInteraction(f_y, f_x, a_x ∩ f_y. area, l)
        else
            SolveInteraction(f_x, f_y, a_x ∩ f_y. area, l)
        end if
    end for
// solve an interaction between a pair of terrain features
// f_lose-feature for which the claim is rejected
// f_win-feature for which the claim is granted
// a-disputed area
// l - level of abstraction
SolveInteraction(feature f_lose, feature f_win, area a, level l):
    // determine whether connection can and should be formed
    if connectionDefined(f_lose.type, f_win.type)
    priority_connect(f_lose, f_win, a, l) > priority_conflict(f_lose, a, l) then
        // interaction is solved with a connection
        f_lose.connectTo(f_win, a, l)
    else
        // interaction is solved by restructuring the losing feature
        f_lose.restructure(a, l)
    end if
```

For each f_y interacting with feature f_x on an area a and on a feature level l, the claim priorities for f_x and f_y are compared, resulting in either f_x losing the claim to f_y or vice versa.

Each interaction is then resolved either by forming a connection to the winning feature f_{win} or, if that is unavailable or has a lower priority, by a conflict where the losing feature f_{lose} has to restructure itself to avoid the lost area of terrain. More concrete, a connection must be defined between the feature types of f_{lose} and f_{win} and it must have a higher priority over conflict, which may not be the case in specific scenarios where a given connection might be deemed too costly or impractical.

Together, these resolution methods offer a generic and automated way to overcome interactions, while still providing the designer with control through setting and overriding the different types of priorities.

4. Framework implementation and results

To validate our approach we implemented representative terrain features for each of the five terrain layers. Their generation procedures are often based on (combinations of) existing procedural methods, but they have been modified to fit in the framework, i.e. to consider their surroundings and context, and to employ the

claim and modification methods of the interaction resolution scheme of Section 3. These features form a basic set useful for many scenarios, to support other types of scenarios; additional features might be integrated such as railways, lakes, canals, harbors, levees, farming fields, etc. For each of these terrain features, we briefly discuss their procedure and conclude with an example session using all features.

4.1. Landscape layer: elevation and soil

The Landscape layer provides the foundation for placing terrain features. Its generation procedure is detailed in our previous work [43], and is outlined in Algorithm 3. The input is derived from the grid of painted ecotopes (see Fig. 1). The definition of each ecotope includes, among other things, a minimum and maximum elevation and a roughness percentage, describing how rough or smooth its terrain should be. From this definition, each cell in the ecotope grid is assigned a randomized, *local* variation of these ranges. These local values in the grid are smoothed using a small Gaussian kernel, to obtain natural changes in elevation. For each point (x, y) in the landscape, the coarse grid g is interpolated with Catmull–Rom splines to obtain the ranges $\overline{r}$ at the desired spatial resolution (e.g. 1 m). Note that the coordinate (x, y) is first perturbed in 2D to decrease the regularity of this interpolation, resulting in (x',y'). A combination of several "flavours" of fractal noise, such as ridged multi-fractal noise [45], mixed according to the roughness factor, are used to determine the elevation value within the range $\overline{r}$. The distribution of soil material is based on the ecotope value at (x',y'), and by mapping the elevation value to a lookup table. The procedure results in $(result_{elevation}, result_{soil})$ being stored at position (x, y) in a height-map data-structure.

Algorithm 3. Landscape layer generation method.

```
// g-coarse grid of ecotopes (elevation ranges, roughness, etc.)
GenerateLandscape(ecotope grid g, random seed s):
  for y = 0 to height do
    for x = 0 to width do
      // perturb x and y locally
      p = perturb(x, y, s)// in range [−π ... π]
      x' = x + offset_perturb cos(p), y' = y + offset_perturb sin(p)
      // obtain 4x4 grid region for Catmull–Rom interpolation
      x_g = (x' − gridcelldim(g)/2)/ gridcelldim(g)// grid x
      y_g = (y' − gridcelldim(g)/2)/ gridcelldim(g)// grid y
      x_1 = ⌊x_g⌋, x_f = x_g − x_1, x_0 = x_1 − 1, x_2 = x_1 + 1, x_3 = x_2 + 1
      y_1 = ⌊y_g⌋, y_f = y_g − y_1, y_0 = y_1 − 1, y_2 = y_1 + 1, y_3 = y_2 + 1
      // interpolate 4x4 grid cells based on fractions (x_f, y_f)
      // interpolation result r̄.xyz = (min, max, roughess)
      r̄ = spline (g, x_0, x_1, x_2, x_3, y_0, y_1, y_2, y_3, x_f, y_f)
      //generate 3 noise values, mix based on roughess
      n̄ = noiseValues (x', y', s) // all in range [−1 ... 1]
      f̄ = mixFactors(r̄.z)// all in range [0 ... 1], |f̄| = 1
      v = f̄ · n̄// combined noise value
      // result for x, y: elevation and soil values
      result_elevation = r.x + (1/2 + v/2)(r.y − r.x)
      result_soil = distribute(x', y', result_elevation, ecotope(g, x_g, y_g))
    end for
  end for
```

The advantage of this combination of fractal noise and coarse grid interpolation is that it allows to evaluate any point independently of its neighbors. This makes it efficient to execute the procedure in parallel (see Section 4.7). Furthermore, the elevation and soil maps can be split in conveniently sized tiles, where each tile is computed and stored independently. This allows us to manage even relatively large landscapes at a high spatial resolution (e.g. 2 or 4 m).

4.2. Water layer: river feature

The generation of a river is constrained by the feature specification polyline and by the local elevation profile. The actual river path is determined by iteratively finding a sub-path for each pair of control points $\overline{p}_{src}$ and $\overline{p}_{dst}$ in the specification. Starting from the highest elevated point $\overline{p}_{cur} = \overline{p}_{src}$, each iteration generates n new candidate points $\overline{p}_{can}$ on a circle with range r_{step} at an interval $[\alpha-\alpha_{dev} \ldots \alpha+\alpha_{dev}]$, where α is the angle from $\overline{p}_{cur}$ towards $\overline{p}_{dst}$ and α_{dev} is an angle range of deviation from α (e.g. 36°). Each of these generated candidates is scored according to the following weighted sum of scores:

$$s_{elevation} = \frac{\overline{p}_{cur}.z - \overline{p}_{can}.z}{\|\overline{p}_{can} - \overline{p}_{cur}\|}$$

$$s_{curve} = 1 - \cos^{-1}\left(\frac{\overline{p}_{dst} - \overline{p}_{cur}}{\|\overline{p}_{dst} - \overline{p}_{cur}\|} \cdot \frac{\overline{p}_{can} - \overline{p}_{cur}}{\|\overline{p}_{can} - \overline{p}_{cur}\|}\right) \Big/ \alpha_{dev}$$

$$s_{can} = w_{elevation} s_{elevation} + w_{curve} s_{curve} + w_{feature} s_{feature}$$

The terms $s_{elevation}$ and s_{curve} denote the scores for local elevation difference and river curvature, respectively. The feature score term $s_{feature}$ is negative if the current segment crosses a feature resulting in a conflict that the river feature would lose. If the candidate with the highest score s_{can} does not adhere to the constraint of monotonously decreasing elevation, the elevation value of the candidate is forced to an elevation slightly lower than its predecessor. The determined river bank is claimed for exclusive use, with possible connections with the various types of roads, other rivers and the sea. The concrete river object is created by segmenting a Catmull–Rom spline along the path, with small variations in width according to the local slope. Using a modification request, the elevation profile is locally adapted to fit with the river's bed and bank, and the soil is changed to a more fertile ecotope, where applicable.

The advantage of this procedure is the generation speed and the fact that it closely follows the designer's specification. It generates good results as long as the control points are reasonably well placed, otherwise they can be interactively adjusted.

4.3. Vegetation layer: forest feature

The forest feature is declared by its coarse area, vegetation density and species that occur in this forest. The input area is slightly perturbed using standard noise techniques to obtain natural tree lines. The refined forest area is claimed on the structural level. An iterative procedure based on the method introduced by Deussen et al. [7], which simulates the competition of plants for natural resources as space and water, determines a distribution of trees of different species in this forest (see [43] for specific details on this procedure). Additional input for this procedure is the local elevation profile and soil material, as the definition of a tree species includes preferences for certain soil types, as well as elevation and slope ranges. For each generated tree object, its position is claimed. The forest feature *restructure* operation removes any trees in the areas lost to other features.

4.4. Road layer: primary road feature and road network

A primary road (e.g. highway, interstate) is declared by coarsely specifying its path. Its generation procedure is based on the path plotting algorithm introduced by Kelly et al. [27]. This algorithm

iteratively finds a smooth path between a set of control points of a polyline defined on an elevation map. It prefers an even change in elevation from start to end, while guaranteeing all control points to be visited and the path to deviate only within a limited range from the specification. The algorithm was extended to avoid unacceptably sharp turns and slopes, to connect to existing features, such as rivers, if necessary, and to avoid potential feature conflicts with negative consequences for the road. Although the procedure uses a scoring mechanism to determine a path, it is not optimizing a cost function, as for instance the A*-based path finding method by Galin et al. [26], and therefore is not guaranteed to find an optimal path in all cases. However, our procedure typically runs more interactively while staying close to the coarse path sketched by the designer, thereby providing more fine-grained user control.

Similarly to the river feature procedure explained in 4.2, a path is constructed between subsequent control points by generating and testing intermediate nodes. The partial procedure outlined in Algorithm 4 explains that the scoring of candidates is again composed of three factors: change in elevation, road curvature and crossed terrain features. However, as in [27], an even change in elevation is preferred, whereas the river prefers the steepest slope. Furthermore, the weights reflect the natural preferences of the river ($w_{elevation}=0.7$, $w_{curve}=0.1$, $w_{feature}=0.2$) versus the road ($w_{elevation}=0.35$, $w_{curve}=0.15$, $w_{feature}=0.5$).

Algorithm 4. Iterative path planning for the road feature.

for all subsequent points $(\overline{p}_{src}, \overline{p}_{dst})$ in $\overline{p}_{start} \dots \overline{p}_{end}$ **do**
  while !reached **do**
    c=generateCandidates($\overline{p}_{src}$, n_{can}, r_{step}, α, α_{dev})
    for all $\overline{p}_{can}$ in c **do**
      // s_e: *local elevation change ratio/global change ratio*
      $s_e = 1-(\frac{|\overline{p}_{src}.z-\overline{p}_{can}.z|}{\|\overline{p}_{dst}-\overline{p}_{src}\|-\|\overline{p}_{dst}-\overline{p}_{can}\|} - \frac{|\overline{p}_{end}.z-\overline{p}_{start}.z|}{\|\overline{p}_{end}-\overline{p}_{start}\|})$
      // s_c: *angle deviation from straight road*
      $s_c = 1-\arccos(\frac{\overline{p}_{dst}-\overline{p}_{cur}}{\|\overline{p}_{dst}-\overline{p}_{cur}\|} \cdot \frac{\overline{p}_{can}-\overline{p}_{cur}}{\|\overline{p}_{can}-\overline{p}_{cur}\|})/\alpha_{dev}$
      // s_f: *lost conflict(s)* $=-\infty$,*connection(s)* $=0$, *none* $=1$
      s_f=score(findInteractions($\overline{p}_{src}$, $\overline{p}_{can}$))
      $s[\overline{p}_{can}] = w_e s_e + w_c s_c + w_f s_f$
    end for
    $\overline{p}_{best}$ = selectCandidate(c, s)
    if $\|\overline{p}_{dst}-\overline{p}_{best}\| \le r_{snap}$ **then**
      // *snap to destination control point*
      reached = true, $\overline{p}_{best} = \overline{p}_{dst}$
    end if
    $\overline{p}_{src} = \overline{p}_{best}$
  end while
end for

After the 3D path has been planned, the elevation profile of this path is fit to a defined slope constraint and smoothed using a Gaussian smoothing kernel. Catmull–Rom interpolation results in a 3D spline. By segmenting and sweeping along the road spline, we obtain the road surface. The area of the road is then claimed on the structural level. Connections have been defined with other road features (junction), river features (bridges), and city features (junctions with local road network's outer ring). Using a modification request, the elevation profile is cut and filled to match with the desired embankment profile of this road. This embankment profile consists of the inner road area, the embankment and the verge. For the verge, a blending zone is established to create a natural smooth transition between the road and the surrounding landscape.

A city's road network consists of secondary and tertiary roads. It is composed of road patterns, as described by Sun et al. [8], where the secondary roads follow the population centers and the tertiary roads follow the grid, radial or mixed patterns. The procedure iteratively places new roads within the city, while checking validity constraints concerning angle, slope and length, and applying snapping rules, such as snapping a new road to existing nearby junctions (see, e.g. [9]). The primary road and a city's road network can form connections by road junctions.

4.5. Urban layer: city feature

A designer defines a city feature by coarsely sketching its outline and selecting a template describing its historical background, which defines how the city core is structured and includes a statistical distribution of types of districts typically found in these kinds of cities. Template examples for a Western European city include the mercantile, the feudal or the absolutistic historic city. The designer can also declare the city's population size, affecting the number and type of residential districts that will be created.

The generation procedure is described in detail in [46]. The city area is first split in one or more clusters. A cluster is an area of landscape limited by either the city bounds or land that is either unsuitable for building or could not be claimed from other features. A distribution of districts within these clusters is determined by an iterative district placement procedure, which uses constraints derived from established models of urban land use. In this procedure, terrain features have an attraction or repulsion influence on districts, depending on their type (commercial, heavy industry, high-class residential, etc.). For each district, candidate locations are generated within the city bounds and evaluated according to a suitability score, which is a weighted sum of scores for several factors, including the type and proximity of other, already placed districts, the location's soil material, the area within the city model (e.g. center, suburban), and the distance to nearby rivers and to primary roads. Each district is placed at the most suitable candidate location.

After districts have been placed, secondary roads and streets are placed to form the road network of the city (see Section 4.4). Within this network, parcels and building lots result from subdividing the available open space, and buildings are generated based on footprint shapes and district types (see [47] for details). Each generated street and building within the city issues a modification request to constrain the local elevation profile.

4.6. Example

To give a more vivid impression of how one can currently design a virtual world in SketchaWorld, we present the (intermediate) results of an example session (see Fig. 4), in which a designer creates, in a couple of minutes, a landscape with a city along a river. The example session also involves several instances of consistency maintenance, resulting in conflicts, connections and modifications to the landscape. Throughout this session, a 3D geometric model derived from the layered semantic virtual world model, is incrementally updated (Fig. 1 right hand), allowing designers to preview the results in 3D.

Fig. 4a shows the basic landscape, sketched in *landscape mode* by brushing the ecotope grid: a coastline with some mountains and dry land in the east. On top of this landscape, using the tools of the *feature mode*, features of the landscape are added. The designer specifies several forest features using polygons, resulting in the vegetation distribution shown in Fig. 4b. Note that the trees do not grow on steep slopes, rocky terrain and barren land.

A river feature is sketched to run from the mountain lands in the south-east towards the ocean. After the river's path is coarsely defined, a suitable course is plotted through the landscape. Through the default priorities, the claim for land is granted to

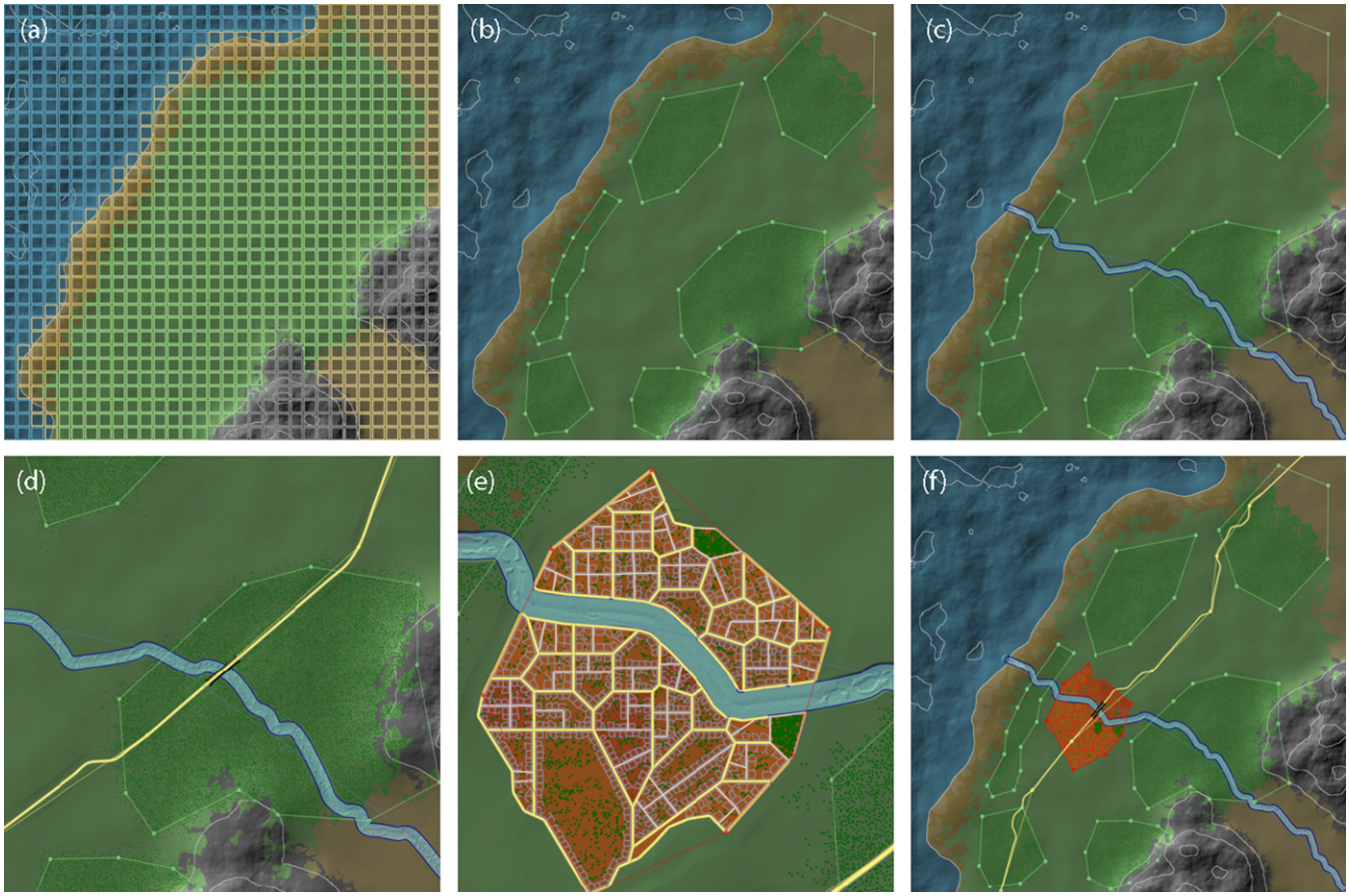

Fig. 4. A procedural sketching session: (a) basic landscape, defined by brushing ecotopes, (b) several forest features, (c) river flowing towards the sea, (d) road feature crossing this river, (e) city created along the river and (f) road rerouted to run through the city.

Fig. 5. 3D virtual world, resulting from the example session of Fig. 4: (a) natural environment with road crossing river, (b) final virtual world and (c) city close-up.

the river, which leads to restructuring of the affected forests. Using a modification request, the river bed and banks are carved into the elevation profile (Fig. 4c). In Fig. 4d the designer adds a main road to the virtual world. Although here the river has a higher priority than the road, a connection is defined between the two features: a bridge is inserted at the river crossing connecting the road segments. Again, the landscape is modified, in this case to form a road embankment (except at the crossing which is still claimed by the river). The forests loose again some of their claimed terrain, now to this primary road. Fig. 5a shows the resulting virtual world.

Subsequently, the designer specifies a small city with a fairly low priority. As a result, its districts and secondary roads have to form around the river, as shown in Fig. 4e. Finally, the designer decides to reroute the primary road to now run through this city (Fig. 4f). This edit action affects the city's claim, therefore, this city is restructured to include this main road and its bridge. Fig. 5b shows the final virtual world, and details of the city are shown in Fig. 5c.

4.7. Implementation and performance

SketchaWorld is implemented in C# and C++ and uses OpenSceneGraph for the visualization of the resulting 3D virtual world. Several of its generation and integration procedures are highly suitable for a parallel implementation, and where therefore implemented on the GPU using CUDA, a C-like programming language for performing general purpose parallel computations

on GPU's. These GPU procedures are an order of magnitude more efficient than their CPU counterparts, thus making interactive procedural sketching feasible. Examples of procedures implemented in CUDA include the elevation and soil map generation procedure, and the procedures for landscape modification requests. In addition, it is used for creating images, such as textures and the 2D view on the virtual world.

Regarding the quality of the resulting 3D virtual world models, we have aimed at achieving *functional realism*, i.e. to generate a plausible and consistent structure and a logical placement of objects and features. Our goal is to be able to export created virtual worlds to a number of engines which differ very much in their graphical capabilities and application (e.g. entertainment games, training simulators). Therefore, instead of focusing on graphics quality, we have concentrated on making new exports straightforward to develop, by clearly separating the virtual world's semantic model from any derived representations (see Fig. 1) and, particularly, by facilitating the replacement of currently used content (e.g. textures) with high-quality content. Currently supported export formats are OpenSceneGraph, COLLADA and GIS raster and vector data; we are developing additional exporters to widen the possible application of our prototype.

The current prototype's performance is good for small and medium sized landscapes (up to 1000 km^2). Most procedural operations typically take between 100 ms and 3 s. However, when modeling larger landscapes continuously updating the 3D geometry of the virtual world becomes a bottleneck that hinders the interactive feedback loop. To further improve the user experience of the prototype and to enable modeling larger landscapes, we plan to optimize some of the complex generation procedures and make more use of GPU computing.

5. Discussion

The combination of procedural sketching and virtual world consistency maintenance helps designers to concentrate on what they want to create and to express this intent at a high level of abstraction, without being bothered with low-level modeling tasks. We have had many informal user sessions with simulation and game development professionals, training instructors, game researchers and students, which provided valuable feedback on our modeling approach and current prototype. Typically, they found the procedural sketching approach very easy to use, requiring no 3D modeling experience, and were able to create a virtual world matching their intent within minutes.

Although the approach gives designers proper control at a high level, especially game designers commented that they miss even more fine-grained user control, for instance to tune generated objects (e.g. individual buildings or trees) to precisely match their intent or artistic vision. We therefore believe that the integration of both procedural generation and manual editing operations is a very promising direction for this research. It seems to combine the best of two worlds, but so far the topic is as good as unaddressed, and it provides numerous new challenges to overcome (see [48]). For instance, manually rerouting a generated street through a city center might entail automatically moving and removing certain blocks of buildings and possibly creating new lots and individual buildings. However, during partial regenerations of such areas (for instance because of feature interactions), manual changes should be preserved as much as possible, which makes these situations especially complex.

Similarly, although designers can influence the consistency maintenance mechanism through setting priorities, either per feature type, or by overriding e.g. the claim priority of a specific feature instance (e.g. a historical wood within a city), we think this level of control may be too coarse to accommodate all modeling scenarios. We are investigating more fine-grained mechanisms for constraining the automatic consistency maintenance, which might include locking features or objects from modification, grouping objects together, introducing local geometric constraints, etc.

A major advantage of this approach is the *flexible* support for integrating both existing and new procedural methods within the same framework, especially since the semantics and, particularly, the interactions of terrain features are defined *independently* from their generation procedures. The framework provides a clean interface and a structured way to incorporate new procedural methods for terrain features as e.g. railways, lakes, canals, or replace existing methods with sophisticated procedures for e.g. complex urban environments. The interaction resolution mechanism is flexible enough to accommodate for interactions with newly introduced features, as well as new types of connections. However, it is primarily geared towards solving interactions that are either user-intended (e.g. a bridge connection) or unavoidable (e.g. removing all trees in the highway path). Naturally, to be integrated effectively in this framework, a procedure has to be made aware of its surroundings and be able to cope with losing claims. Furthermore, the procedure should be made compatible with interactive procedural sketching, which might not be straightforward for procedural approaches that are based on optimization or growth. For these procedures, a significant amount of work could be required to fit them in the framework, after which the respective feature interactions can be properly handled.

The additional execution time for consistency maintenance is in typical modeling cases hardly noticeable. However, generating or removing a large, high priority feature can, of course, have a significant impact on nearby features, resulting in a cascade of many conflicts and connections to be resolved. Such extreme situations can be alleviated by using proper heuristics or conflict avoidance during procedural generation (see e.g. Section 4.4).

The framework SketchaWorld is currently used in several research projects and for a number of simulators. We are investigating possibilities of making it available to a wider audience.

6. Conclusions

We identified the challenges currently faced by designers of virtual worlds, arising from limitations of both manual and procedural modeling techniques. From recent developments in procedural modeling research, we concluded that improvements in user control, interactivity and integration of results are now not only feasible but also essential to increase the acceptance of procedural techniques in mainstream virtual world development.

In this article we introduced declarative modeling of virtual worlds, a novel approach that combines the integrated use of various procedural modeling techniques with a semantics-driven model to capture designer's intent. This approach was illustrated using our research prototype, the modeling framework SketchaWorld. One of its main contributions is *procedural sketching*, which combines user interaction, data representation and procedural techniques in order to enable designers of virtual worlds to concentrate on stating *what* they want to create, instead of on describing *how* they should model it. This is achieved by means of familiar sketch actions, promoting interactive experimentation to quickly see the effects of procedural modeling operations.

This interaction method is complemented by *virtual world consistency maintenance*, which automatically solves conflicts due to spatial interactions occurring between terrain features and their surroundings. By maintaining in the virtual world semantic model more information than simply its geometric data, this technique assists designers in keeping it in a coherent state, in which the semantics of all terrain features is preserved. This

encourages designers to freely experiment with design alternatives without the penalty of laborious manual editing, as feature integration and consistency are taken care of automatically. In traditional modeling systems, in turn, making large scale changes to a virtual world is mostly prohibitive, in terms of both time and manual effort.

Current research is focusing on performance optimization of individual procedures and on enhancements of the quality and diversity of the 3D output. However, our main research challenge ahead concerns new mechanisms for designers to interactively manipulate generated objects, and for controlling and handling the consequences of these manual operations. We believe that supporting such a fine level of editing will be crucial to have designers fully profit from procedural techniques, while remaining in control over their outcome.

Concluding, the *declarative modeling approach* presented here provides virtual world *designers* with the productivity gain of procedural generation techniques, while still allowing for user control; it provides *researchers* with a flexible platform to integrate new techniques from procedural modeling research within an interactive environment; and, finally, it provides whole new groups of *non-specialists* with a clear and accessible method to create complete 3D virtual worlds in minutes.

Acknowledgements

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research and the Netherlands ICT Research and Innovation Authority. We would like to thank the reviewers for their constructive criticism.

References

[1] Miller GSP. The definition and rendering of terrain maps. In: SIGGRAPH '86: proceedings of the 13th annual conference on computer graphics and interactive techniques, vol. 20. New York, NY, USA: ACM; 1986. p. 39–48.
[2] Musgrave FK. Methods for realistic landscape imaging. PhD thesis, Yale University, New Haven, CT, USA; 1993.
[3] Kelley AD, Malin MC, Nielson GM. Terrain simulation using a model of stream erosion. In: SIGGRAPH '88: proceedings of the 15th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1988. p. 263–8.
[4] Prusinkiewicz P, Hammel M. A fractal model of mountains with rivers. In: Proceeding of graphics interface '93, 1993. p. 174–80.
[5] Teoh ST. River and coastal action in automatic terrain generation. In: CGVR 2008: proceedings of the 2008 international conference on CG & VR. Las Vegas, Nevada, USA: CSREA Press; 2008. p. 3–9.
[6] Prusinkiewicz P, Lindenmayer A. The algorithmic beauty of plants. New York, NY, USA: Springer-Verlag; 1990.
[7] Deussen O, Hanrahan P, Lintermann B, Měch R, Pharr M, Prusinkiewicz P. Realistic modeling and rendering of plant ecosystems. In: SIGGRAPH '98: proceedings of the 25th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1998. p. 275–86.
[8] Sun J, Yu X, Baciu G, Green M. Template-based generation of road networks for virtual city modeling. In: VRST '02: proceedings of the ACM symposium on virtual reality software and technology. New York, NY, USA: ACM; 2002. p. 33–40.
[9] Parish YIH, Müller P. Procedural modeling of cities. In: SIGGRAPH '01: proceedings of the 28th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 2001. p. 301–8.
[10] Wonka P, Wimmer M, Sillion F, Ribarsky W. Instant architecture.In: SIGGRAPH '03: proceedings of the 30th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 2003. p. 669–77.
[11] Müller P, Wonka P, Haegler S, Ulmer A, Gool LV. Procedural modeling of buildings. In: SIGGRAPH '06: proceedings of the 33rd annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 2006. p. 614–23.
[12] Finkenzeller D. Detailed building facades. IEEE Computer Graphics and Applications 2008;28(3):58–66.
[13] Smelik RM, de Kraker KJ, Tutenel T, Bidarra R, Groenewegen SA. A survey of procedural methods for terrain modelling. In: Proceedings of the CASA workshop on 3D advanced media in gaming and simulation (3AMIGAS), Amsterdam, The Netherlands, 2009.
[14] Stachniak S, Stuerzlinger W. An algorithm for automated fractal terrain deformation. Computer Graphics and Artificial Intelligence 2005;1: 64–76.
[15] Zhou H, Sun J, Turk G, Rehg J. Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics 2007;13(4): 834–48.
[16] Saunders RL, Terrainosaurus: realistic terrain synthesis using genetic algorithms. Master's thesis, Texas A&M University; December 2006.
[17] Kamal KR, Uddin YS. Parametrically controlled terrain generation. In: GRAPHITE '07: proceedings of the fifth international conference on computer graphics and interactive techniques in Australia and Southeast Asia. New York, NY, USA: ACM; 2007. p. 17–23.
[18] Belhadj F. Terrain modeling: a constrained fractal model. In: AFRIGRAPH '07: proceedings of the fifth international conference on computer graphics, virtual reality, visualisation and interaction in Africa. New York, NY, USA: ACM; 2007. p. 197–204.
[19] Doran J, Parberry I. Controlled procedural terrain generation using software agents. IEEE Transactions on Computational Intelligence and AI in Games 2010;2(2):111–9.
[20] Schneider J, Boldte T, Westermann R. Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In: Vision, modeling and visualization 2006, 2006.
[21] Stava O, Beneš B, Brisbin M, Křivánek J. Interactive terrain modeling using hydraulic erosion. In: EG SIGGRAPH symposium on computer animation. Dublin, Ireland: Eurographics Association; 2008. p. 201–10.
[22] Gain J, Marais P, Strasser W. Terrain sketching. In: I3D '09: proceedings of the 2009 symposium on interactive 3D graphics and games. New York, NY, USA: ACM; 2009. p. 31–8.
[23] de Carpentier G, Bidarra R. Interactive GPU-based procedural heightfield brushes. In: Proceedings of the fourth international conference on the foundations of digital games, Florida, USA, 2009.
[24] Chen G, Esch G, Wonka P, Müller P, Zhang E. Interactive procedural street modeling. In: SIGGRAPH '08: proceedings of the 35th annual conference on computer graphics and interactive techniques, vol. 27. New York, NY, USA: ACM; 2008. p. 1–10.
[25] McCrae J, Singh K. Sketch-based path design. In: GI '09: proceedings of graphics interface 2009. Toronto, Ontario, Canada: Canadian Information Processing Society; 2009. p. 95–102.
[26] Galin E, Peytavie A, Marchal N, Gurin E. Procedural generation of roads. In: Computer graphics forum: eurographics 2010 proceedings, vol. 29. Norrköping, Sweden: Eurographics Association; 2010. p. 429–38.
[27] Kelly G, McCabe H. Citygen: an interactive system for procedural city generation. In: Proceedings of the fifth annual international conference in computer game design and technology, Liverpool, UK, 2007. p. 8–16.
[28] de Villiers M, Naicker N. A sketching interface for procedural city generation. Technical Report, University of Cape Town; November 2006.
[29] Weber B, Müller P, Wonka P, Gross M. Interactive geometric simulation of 4D cities. Proceedings of Eurographics 2009;28:481–92.
[30] Vanegas CA, Aliaga DG, Beneš B, Waddell PA. Interactive design of urban spaces using geometrical and behavioral modeling. ACM TOG: proceedings of ACM SIGGRAPH Asia, vol. 28(5), 2009. p. 1–10.
[31] Lipp M, Wonka P, Wimmer M. Interactive visual editing of grammars for procedural architecture. In: SIGGRAPH '08: proceedings of the 35th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 2008. p. 1–10.
[32] Amburn P, Grant E, Whitted T. Managing geometric complexity with enhanced procedural models. In: SIGGRAPH '86: proceedings of the 13th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM Press; 1986. p. 189–95.
[33] Bruneton E, Neyret F. Real-time rendering and editing of vector-based terrains. In: Computer graphics forum: eurographics 2008 proceedings, vol. 27, Crete, Greece, 2008. p. 311–20.
[34] Tutenel T, Bidarra R, Smelik RM, de Kraker KJ. The role of semantics in games and simulations. ACM Computers in Entertainment 2008;6:1–35.
[35] Bidarra R, Bronsvoort W. Semantic feature modelling. Computer-Aided Design 2000;32(3):201–25.
[36] Kallmann M, Thalmann D. Modeling objects for interaction tasks. In: EG workshop on animation and simulation, 1998. p. 73–86.
[37] Tutenel T, Smelik RM, Bidarra R, de Kraker KJ. Using semantics to improve the design of game worlds. In: Proceedings of AIIDE 2009—fifth conference on artificial intelligence and interactive digital entertainment, Stanford, CA, USA, 2009.
[38] Tutenel T, Smelik RM, Bidarra R, de Kraker KJ. A semantic scene description language for procedural layout solving problems. In: Proceedings of AIIDE 2010—sixth conference on artificial intelligence and interactive digital entertainment, Stanford, CA, USA, 2010.
[39] Torpy A. L3DT 〈http://www.bundysoft.com/L3DT/〉.
[40] Greenworks, XFrog 〈http://www.xfrog.com〉.
[41] Procedural, inc., CityEngine 〈http://www.procedural.com〉.
[42] PixelActive, Cityscape 1.8 〈http://pixelactive3d.com〉.
[43] Smelik RM, Tutenel T, de Kraker KJ, Bidarra R. Declarative terrain modeling for military training games. International Journal of Computer Game Technology (IJCGT) 2010;2010:1–11.
[44] Smelik RM, Tutenel T, de Kraker KJ, Bidarra R. Interactive creation of virtual worlds using procedural sketching. In: Proceedings of eurographics 2010: short papers. Eurographics Association; 2010.

[45] Musgrave FK, Kolb CE, Mace RS. The synthesis and rendering of eroded fractal terrains. In: SIGGRAPH '89: proceedings of the 16th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1989. p. 41–50.
[46] Groenewegen SA, Smelik RM, de Kraker KJ, Bidarra R. Procedural city layout generation based on urban land use models. In: Eurographics 2009: short papers. Munich, Germany: Eurographics Association; 2009. p. 45–8.
[47] Smelik RM, Tutenel T, de Kraker KJ, Bidarra R. A case study on procedural modeling of geo-typical Southern Afghanistan terrain. In: Proceedings of the IMAGE 2009 conference. St. Louis, MO, USA: IMAGE Society; 2009. p. 329–37.
[48] Smelik RM,Tutenel T, de Kraker KJ, Bidarra R. Integrating procedural generation and manual editing of virtual worlds. In: PCGames '10: proceedings of the 2010 workshop on procedural content generation in games. New York, NY, USA: ACM; 2010. p. 1–8.