3D Game Model and Texture Generation Using Interactive Genetic Algorithm

DUMIM YOON and KYUNG-JOONG KIM, Sejong University

Recently, the production of big games is expensive, and it is very difficult to gain attention from players. Although gamers' expectations are very high, game companies' resources are limited to maximize the quality of the games. Mod (game content editing) can be one of the solutions to this problem. It allows gamers satisfy themselves by creating and sharing their Mod. It increases the gamers' playing time and sales. However, few users have the knowledge and special ability to create Mods, so most gamers just use a Mod made by someone else. In this article, we propose a method to generate contents for 3D game objects and textures. Our method enables the construction of a three-dimensional object using the grown building footprints by the L-system or the combination of polygons, and it also provides a web-based interactive genetic algorithm interface to users for changing shapes. Furthermore, it also provides a function to evolve various texture images from an original texture file. We demonstrated the possibility of our systems using The Open Racing Car Simulator (TORCS). These results reveal that three-dimensional objects and textures can be created from our method by amateur users.

CCS Concepts: • **Human-centered computing** → *User interface design*; • **Computing methodologies** → *Mesh geometry models*

Additional Key Words and Phrases: 3D Model, game contents generation, interactive generic algorithm

ACM Reference Format:
Dumim Yoon and Kyung-Joong Kim. 2017. 3D game model and texture generation using interactive genetic algorithm. Comput. Entertain. 14, 1, Article 1 (January 2017), 16 pages.
DOI: http://dx.doi.org/10.1145/2735382

1. INTRODUCTION

The video game industry has successfully expanded their territory into personal computer, video game consoles, and mobile devices. It is common that games include fantastic graphics, multi-modal natural interfaces, and social network play. As a result, players usually have very high expectations, but the development cost for the game increases rapidly. One of the most expensive parts in the game development is contents creation by designers. To reduce the cost, recently, there have been works on procedural game content generation, which uses algorithmic procedure to create game contents [Togelius et al. 2011]. In the line of research, it is important to personalize game contents in the automatic generation. Recently, game users have great interest to edit games, and companies have started to supports the game modification [Bostan and Kaplancali 2010]. It opens the door to the gamers for the creation of their own

Ruck Thawonmas, Guest Editor
This research was supported by Basic Science Research Program and the Original Technology Research Program for Brain Science through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0012876) (2010-0018948).
Authors' addresses: D. Yoon and K.-J. Kim, University of Sejong; emails: krad@hanmir.com, kimkj@sejong.ac.kr.

DOI: http://dx.doi.org/10.1145/2735382

personal games by changing the shapes, textures, and rules of the games. For example, game developers create only parts of the planned contents and allow the gamers who were dissatisfied on the original contents to create extra contents by themselves. This approach, called Game Mod, usually requires high-level skills and semi-expert knowledge [Bostan and Kaplancali 2010].

Especially, to create three-dimensional (3D) game contents has been a high barrier for amateurs. Therefore, in this article, we propose 3D game contents generation based on an interactive genetic algorithm (IGA) that creates novel contents based on users' feedback. It evolves the 3D shapes of buildings and textures for the structure. The textures are evolved by our photo retouching system [Yoon and Kim 2012]. It allows amateurs to make custom 3D game contents for editable games.

2. RELATED WORKS

In this section, we introduce game content generation works based on evolutionary computation. The purpose of the game contents generation using evolution is to find novel and personalized contents automatically. Usually, they adopt the IGA because the evaluation of the solution is subjective and a human should be involved in the search process. Loiacono et al. [2011] used the IGA to generate novel racing tracks for a simulated car racing game (http://trackgen.pierlucalanzi.net/). This method generates novel game tracks based on anonymous users' evaluation (positive or negative) counts via the web. It is possible to download the evolved tracks and applied them into the game.

Hastings and Stanley [2010] evolved a bullet pattern using artificial neural networks in their shooting games (Galactic Arms Race) (http://gar.eecs.ucf.edu/). Their results show that it is possible to automatically create interesting and successful bullet patterns. It can save the effort for game developers to design effective and interesting attack patterns.

Hastings et al. [2009] evolved a particle system for computer graphics and animation. This method can make customized particle motions for graphics or games through interactive evolutionary computation.

Eisenmann et al. [2011] designed character animations and particle systems by evolution (http://accad.osu.edu/Projects/Evo/).

All of these methods have something in common: the solution is too difficult for amateurs and determined by personal sense.

3. L-SYSTEM–BASED SIMPLE 3D BUILDING MODEL GENERATION

Although background buildings are minor content compared to characters, they are definitely needed in urban background games. It is important to increase the diversity of buildings in the background to improve the user's experience. In this section, we introduce the automatic building structure evolution.

3.1. Making a Building Footprint Using the L-System

In this article, we generate an arbitrary building shape by growing a footprint of the building (Figure 1). Although there are some works on automatic building design system [Müller et al. 2006; Wonka et al. 2003], we focus on using IGA for game content. To represent the footprint, we adopt the L-system [Prusinkiewicz and Lindenmayer 2012], which is a powerful tool to generate complex shapes using simple grammars. It generates a randomized result based on a seed value using a reproducible random number generator (RNG). In this work, we use the RNG Marsaglia [Marsaglia 2003].

Initially, a square building footprint is generated, and it grows to other shapes by the L-system grammatical rules. Although the starting shape can be angular, circular, or polygonal, we test only the square type. The grammatical rule for the L-system is

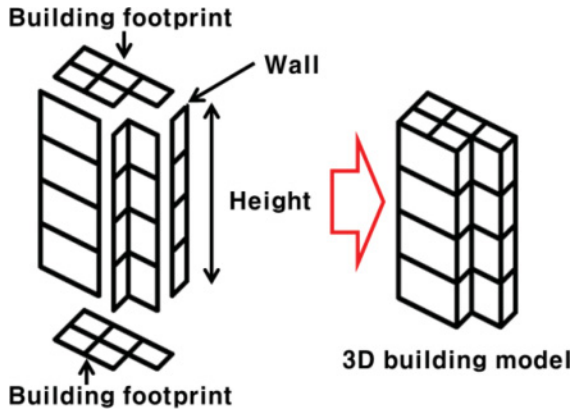
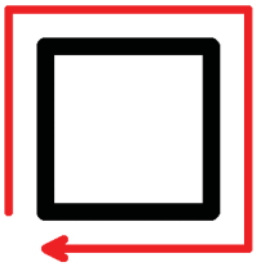


Fig. 1. Structure of 3D building model.



(FRFRFRFR)

Production Rule : "F" → "LFRFRFL"

An Example (stochastic L-system)

"FRFRFRFR" → "FRFRFRLFRFRFLR"

Fig. 2. Initial building footprint (top), grammar rule (middle), and an example (bottom).

written with R, L, and F. These three characters, R, L and F, mean clockwise rotation 90°, –90°, and forward. The R and L just change the direction of the drawing. The F is actually drawing a single unit line to the current direction.

The starting point of our stochastic L-system is defined as a string "FRFRFRFR," which draws a square (Figure 2). The production rule is FLFRFRFL defined manually. In the production, for each "F" in the string, we get a random value ranged from 0 to 1, and if the value is smaller than a predefined probability threshold (randomly chosen), the rule is applied. In the example, only one "F" is replaced with "LFRFRFL."

3.2. Modifying Results

From the grammatical production, we can get a building footprint represented as a string of the alphabet. It shows that "FRRF" or "FLLF" result in a single line instead of polygons. The strings are removed during the production. Besides the two strings, there is redundant information that can be shortened (Table I).

The modification is deterministic, and all the substrings matched to the pattern are replaced with the alternatives.

Table I. Modification on the String

Original String	Replacement
“FLLF”	“LL”
“FRRF”	“RR”
“RRRR”	“”
“LLL”	“R”
“RRR”	“L”
“LL”	“RR”
“RL”	“”
“LR”	“”
“FRFRFRFR”	“L”
“FLFLFLFL”	“R”

3.3. Web-Based Evolution

Because the building designs are for the games, the evaluation on them is subjective. The interactive evolutionary computation is a useful tool to search for solutions based on subjective evaluation. Usually, it maintains the small size of populations, and users are involved in the evaluation. Based on the scores from users, the genetic algorithm creates a new generation from the parents. Because of user fatigue, it is important to get a reasonable solution with a small number of evaluations.

Traditionally, the IGA runs on a personal computer. However, the new web-based environments allow users to evaluate or evolve solutions anywhere and anytime. If users can access the Internet from their smart phone, it is possible to run the evolution with that device. Also, if the evaluation is open to the public, it can search for a solution with collective intelligence. There are several examples exploiting votes from anonymous users. For example, users can evolve 3D objects through a webpage, and they can be printed using 3D printer (http://endlessforms.com).

Our interface runs on the web, and anyone can access to the program to evolve building footprints (http://cilab.sejong.ac.kr/EC).[1] In this work, the chromosome consists of values for a seed for the RNG, the iteration number in the L-system production (0–10), and the height of the building (0–15). The population size is 12. After the initialization of the population, the interface shows 3D buildings generated with the default texture. By clicking the building, the user can give scores and get a corresponding 3D file exportable to 3D-simulated car racing games (e.g., TORCS). The evolution continues by creating a new population based on user evaluation and genetic operations.

If the user selects a building, the score is 1; otherwise, it is 0. It means that the chromosome scored as 0 has no chance to survive in the next generation. Although the individual is not selected, there is a small chance that the parts of the individual's chromosome are useful to create novel solutions. The following equation is used to adjust the original fitness value (in this article, $K = 3$).

$$\mathrm{f_i} = (\mathrm{S_i} - S_w) + \frac{S_b - S_w}{K - 1}, (K > 1)$$

$\mathrm{f_i}$: $\mathrm{i} - \mathrm{thfitness}$

$\mathrm{S_i}$: $\mathrm{i} - \mathrm{thscore}$

$\mathrm{S_b}$: thebestscore

$\mathrm{S_w}$: theworstscore

K: SelectionPressure

[1]Google Chrome, Mozilla Firefox, Safari, and Opera platforms.

(a) Evolved contents

(b) a snapshot in the evolution (Green means the selection by a user)

```
AC3Db
MATERIAL "ac3dmat1" rgb 1 1 1 amb 0.2 0.2 0.2
   emis 0 0 0 spec 0.5 0.5 0.5 shi 10 trans 0
OBJECT world
kids 20
OBJECT poly
name "poly"
loc 0 0 0
texture "building1.rgb"
numvert 4
0 0 0
0 10 0
...
```

(c) An example of a 3D file exported from the interface

Fig. 3. Some snapshots from the interface and the exported 3D contents for games.

The chromosome is presented with 6 bytes (4 bytes for the seed, 1 byte for the number of iterations in the production, and 1 byte for the height). It adopts a standard 1-point crossover. The mutation operator is applied to the chromosome in the byte level. Each byte of the chromosome is tested regardless of whether the mutation is applied with small mutation probability (in this article, 3%).

In previous works, researchers attempted to create 2D contents or single-colored 3D contents [Clune and Lipson 2011; Cardamone et al. 2011; Secretan et al. 2008]. In this work, the interface is used to evolve 3D buildings. Although the texture of the building is not evolved together with the building structure, we also provide separate systems to evolve the texture of the game contents.

The system uses HTML 5 and WEBGL. It allows 3D objects to be rendered in the users' web browsers. The Figure 3(a) shows a set of successful 3D building models and their average score. The interface can export the successful building structure to AC3D format.

4. COMBINATION-BASED MODERN 3D BUILDING MODEL GENERATION

Modern buildings have complex shapes, but some shapes consist of simple shape combinations. In other words, a complex building can be a combination of simple shapes. Figure 4(a) describes this basic idea.

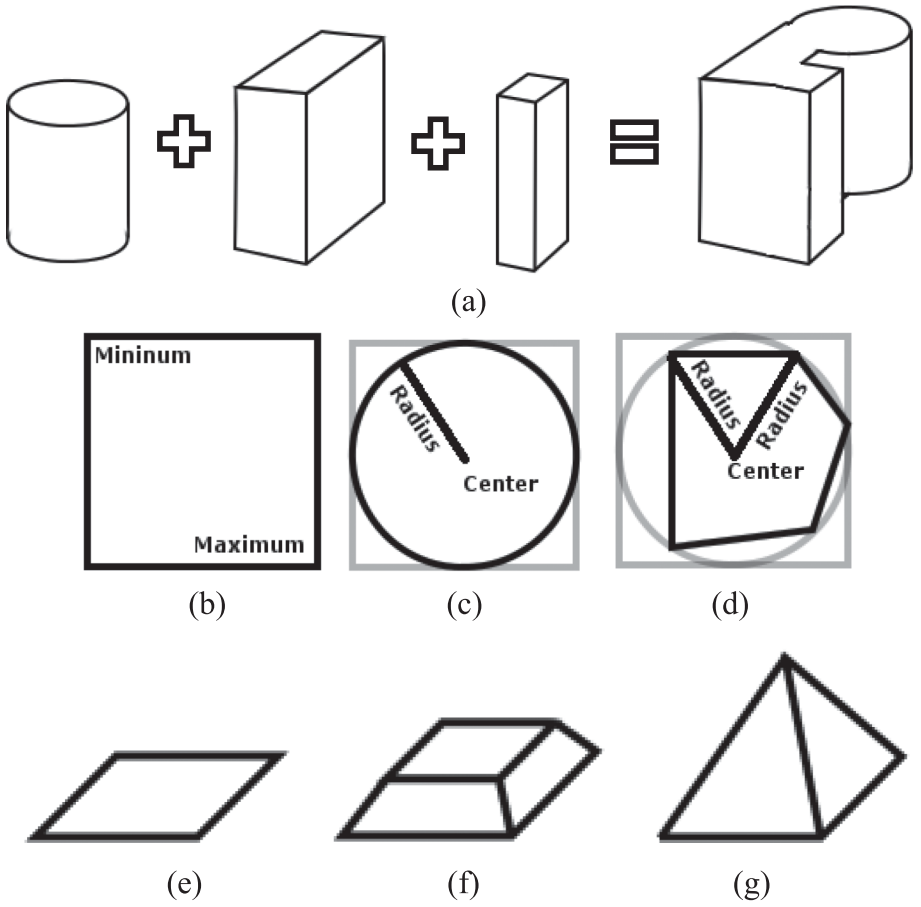


Fig. 4. Top: The complex building made of simple polygons (a). Middle: Top view of generated pillars of rectangle, circle, and polygon and description (b–d) Bottom: Three type of ceiling: flat, bevel, and spire (e–g).

4.1. Generation Method of Components

We set three types—rectangle, circle, polygon-shaped pillar—of components for combination. These three shapes have a convex shape because a convex polygon can easily be handled in a 3D environment, and it can make a concave polygon by combination. And each component should be a gene having the same size. But a polygon is needs more data than a box and a circle. For this problem, we use the reproducible RNG.

A gene has six parts (Table II). "Type" determines the shape of the pillar as rectangle (0), circle (1), or polygon (2, 3). The polygon type has more various changes then we set it to two values. "Cap type" determines three types of ceilings. In our study, it is enough that Positions X and Y and Height are set to just 1 byte.

The 3D model component generating sequence is used different method by each type.

The first type is a rectangle that has simple method. It is completed by getting four numbers from the RNG. Each number corresponds to Minimum *XY*-axis and Maximum *XY*-axis. The second type, circle, is needed to get the radius and center from rectangle points, which can be calculated through subtraction and division. Then it can make a circular polygon with a sine function. The last type of polygon is more complex. The

Table II. Structure of Gene

Name	Size	Value
Type	4 bits	0 : Rectangle 1 : Circle 2,3 : Polygon
Cap type	4 bits	0 : Flat 1 : Bevel 2 : Spire
Position X	1 byte	0 ~ 255
Position Y	1 byte	0 ~ 255
Height	1 byte	0 ~ 255
Generator seed	4 bytes	0 ~ 0x7FFFFFFF
Total	8 bytes	

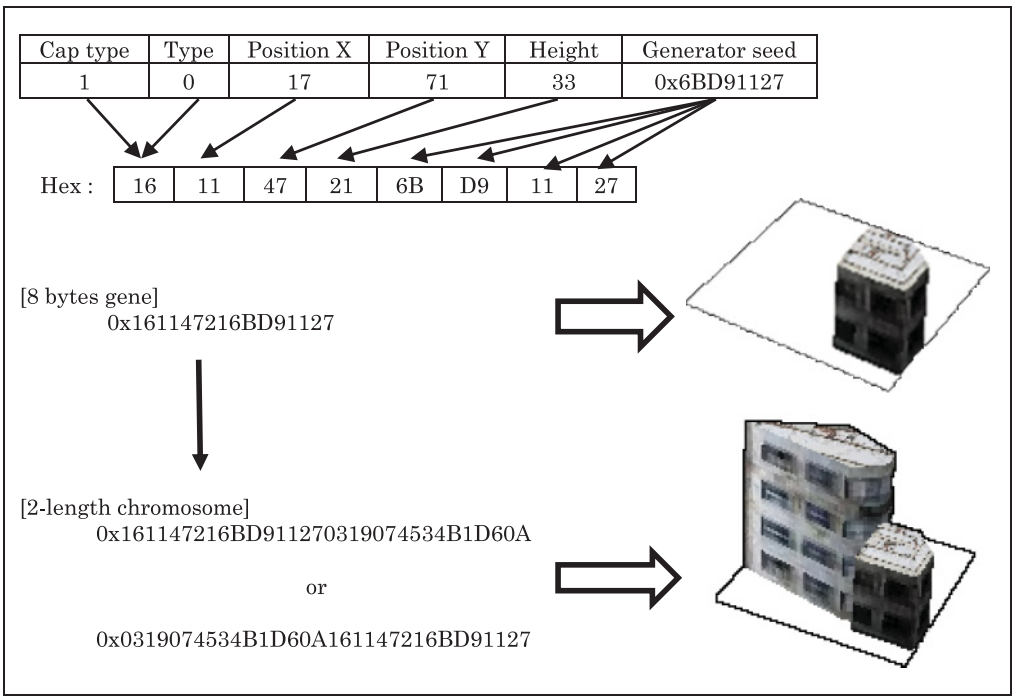


Fig. 5. Structure of variable-length chromosome.

first step is to get the radius and center point value, as was done with the circle, and to get the number of vertices of the new polygon, using the RNG. The next step is to get number between 0 and 2π as the number of vertices. After sorting these numbers, this number list is to be each vertex of clockwise convex polygon. This method useful for genetic algorithm because it needs same size as other types and can make many variations. Figure 4(b–g), Algorithm 1, and Figure 5 show the generating methods of the components.

The pillar, which is generated by the method, will combine with one of three types of ceilings. This ceiling can also be made based on four parameters (minimum, maximum, center, and radius).

ALGORITHM 1: 3D Model Component Generation (Pillar)

Input: seed for random generator (seed), height for building (height), building type (type)
Output: 3D vertex array
(rand function is return to integer number)

```
rand_new_seed(seed)
width = rand()%max_position_x
height = rand()%max_position_y
P1 = {x: (max_position_x-width)/2, y:(max_position_y-height)/2}
P2 = {x: P1.x+width, y: P1.y+height}
coordinate P1, P2 into area(0∼max_position_x, 0∼max_position_y)
switch type
    case rectangle:
      result = Makebox(P1,P2,height)
    case circle:
      center = (P2 – P1)/2
      radius = center.x;
      result = Makecylinder(center,radius,height);
    case polygon:
      center = (P2 – P1)/2
      radius = center.x;
      count = rand ()%max_polygon_number
      Point_array = []
      for 0 to count do
          Point_array.add((rand()%1001)/1000.0 *2 π)
      end
      Sort(Point_array)
      result = Makepolygon(center,radius,count,Point_array,height)
end
return result
```

4.2. Variable-Length Genetic Algorithm for Model Generation

The pillar, which is generated by the method, can change to a gene for a genetic algorithm. In our study, we determined a single gene size to be 8 in accordance with Table II. A variable-length chromosome means a combination of multi-components because a gene is handled as a component.

4.3. Web-Based Evolution

As described previously, a chromosome's size is a multiple of 8. The genetic algorithm operates based on a byte unit in this study. Therefore, we get the byte array through the division of a chromosome. The array size is also a multiple of 8. Most of the evolution algorithm is similar with web-based evolution discussed in Section 3.3.

In this study, the types of operations of the genetic algorithm are crossover, mutation, extension, and reduction. The crossover is to create children by changing an array data based on a crossover point. When the crossover is operated, the crossover point has a range limit with a shorter length than the array in Figure 6. The mutation can change each byte in the array to a random value with low probability. The extension is extending the array by 8 bytes and fills it with random values. Of course, each random value has limit preset maximum size (Table II). The reduction is to reduce 8 bytes from the array when an array has more than 8 bytes. Algorithm 2 is the pseudocode of an evolution step.

Figure 7 shows view of a web-based interface (http://cilab.sejong.ac.kr/EC_01). It can generate a 3D model based on the IGA and all random parameters. If you want get

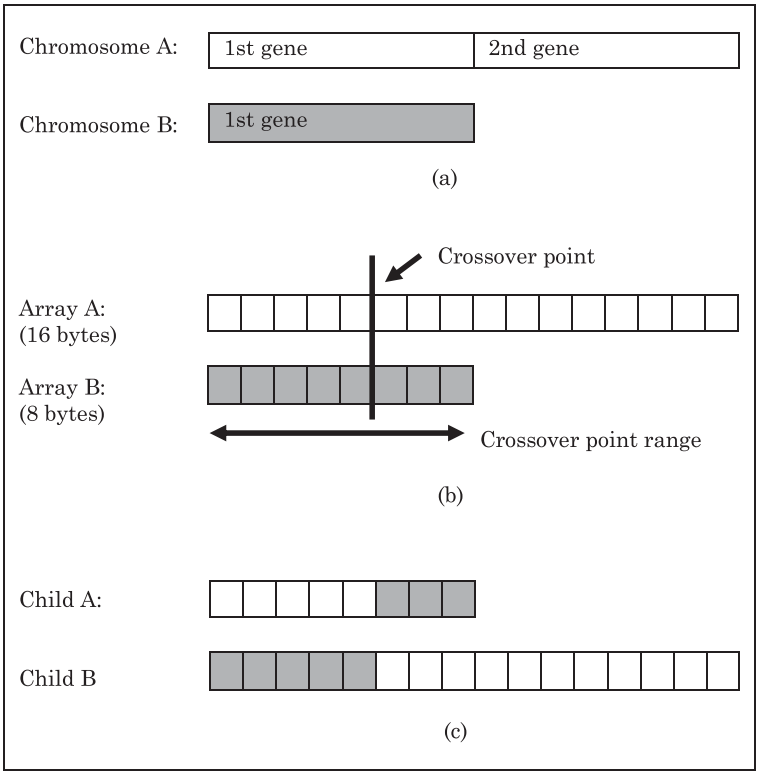


Fig. 6. The method of crossover.

ALGORITHM 2: A Step of Genetic Algorithm Sequence

Input: Population of this generation (population), probability of mutation (P_mutation), probatility of extention (P_extention), probability of reduction (P_reduction)
Output: Population of next generation

```
Array_list = []
foreach chromosome in population do
        Array_list.add( chromosome.toarray() )
end

children_list = []
do
        childA,childB = getCrossoverByFitness(Array_list)
        if not isDuplicate(childA, children_list) then children_list.add(childA)
        if not isDuplicate(childB, children_list) then children_list.add(childB)
while children_list.length < population.size
children_list.resize(population_size)

mutation (children_list, P_mutation)
reduction (children_list, P_reduction)
extension (children_list, P_extension)
```

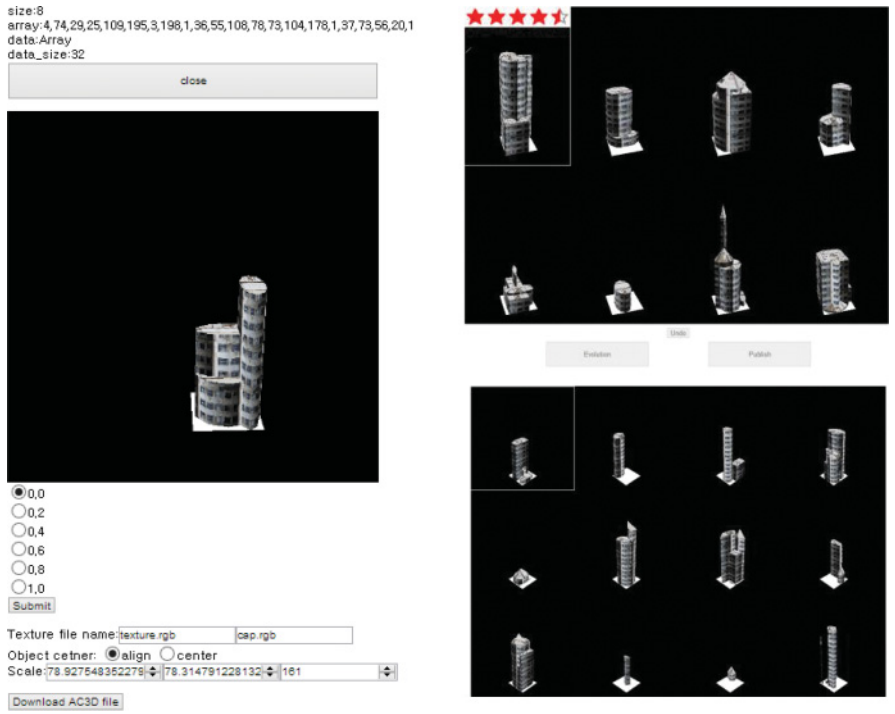


Fig. 7. Left: The building publish screen. Right: Evolved building (top) and web-based evolution (bottom).

some model data, you can download one by clicking on the model or publish button as AC3D format file.

5. TEXTURE RETOUCHING BY IGA

Using the web-based interface, users can generate a 3D structure of the buildings. The interactive evolution searches for the shape of the building's footprint and height. However, in the evolution, the texture of the building is not changed. At this moment, the evolution of the 3D model and its texture is not integrated as a single system.

Yoon and Kim [2012] developed an IGA-based system to evolve a set of filters to be applied to the input image. The interface shows 12 new images filtered from the input. It also evolves the number of filters to be applied to the target. It starts from a single filter, but the genetic operators add or delete filters from the initial filter sets.

Unlike the previous evolution of the 3D model, this evolution assumes that the length of the chromosome is not fixed. The length is proportionate to the number of filters used. For example, a solution can be a sequence of filters (e.g., filter 1, filter 5, filter 6, and filter 3). The filters are applied to the original input image, and the output image is passed to the next filter.

In commercial photo editing tools, they support many filters (smoothing, sharpening, special effects, etc.). However, they are not useful for our filter evolution software because they do not provide an interface to automate the sequential filtering. The GNU Image Manipulation Program (GIMP) supports a script-based interface to process images, and they can be combined with our IGA software. Our system generates a GIMP script file dictating the sequence of filters to be applied to the input images and pass it to the GIMP. The program outputs the results of the sequential filtering.

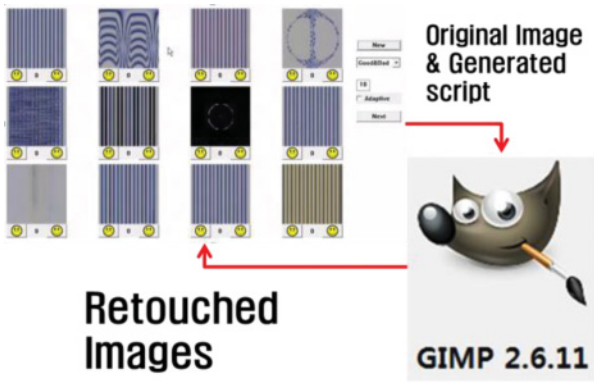


Fig. 8. An overview of the texture retouching system.

```
(define (custom-filterID srcfilename destfilename)
(let* ((image (car (gimp-file-load RUN-NONINTERACTIVE srcfilename
srcfilename)))
(drawable (car (gimp-image-get-active-layer image))))
(if (not (= RGB (car (gimp-image-base-type image)))) (gimp-image-convert-rgb
image))
(gimp-message "processingID")
....
(gimp-image-flatten image)
(gimp-file-save RUN-NONINTERACTIVE image (car ( gimp-image-get-active-layer
image)) destfilename destfilename)
(gimp-image-delete image) ))
```

Fig. 9. A template of script-fu for the filter evolution. Each filter set in the population has a unique ID. In the middle of the script-fu code, a set of codes for filtering is added.

Finally, our program visualizes the number of new images filtered, and users give scores for the goodness of the processing. Usually, it is not easy to create novel images from scratch, but the retouching system can generate a lot of interesting figures because it starts from user-created photos or patterns (Figure 8).

5.1. Frequency Assignment

GIMP is a cross-platform image editor tool. It supports drawing, editing, and special effects and includes various filters. Also, it has an interface to do batch processing, called *script-fu*. Because of the property, the program has been used in several research works [Petcu and Iordan 2006; Bucior and Ventures 2007]. For example, GIMP was used to create artistic images [Valente and Klette 2010], sharpen images [Jaksa et al. 2003], and lighten images [Hara et al. 2009]. In this work, the script-fu (script-based interface for the GIMP) is used to combine a set of filters to produce a new image from an original image (Figure 9).

5.2. Evolution

Figure 10 shows an interface for the image retouching. Initially, it generates 12 filtered images and waits for the user's feedback on them. For each image, the user can express preferences as "good" or "bad." For the "good" decision, the fitness value is 1. Otherwise, the value is 0. It also supports sliding scale input,and five stars.

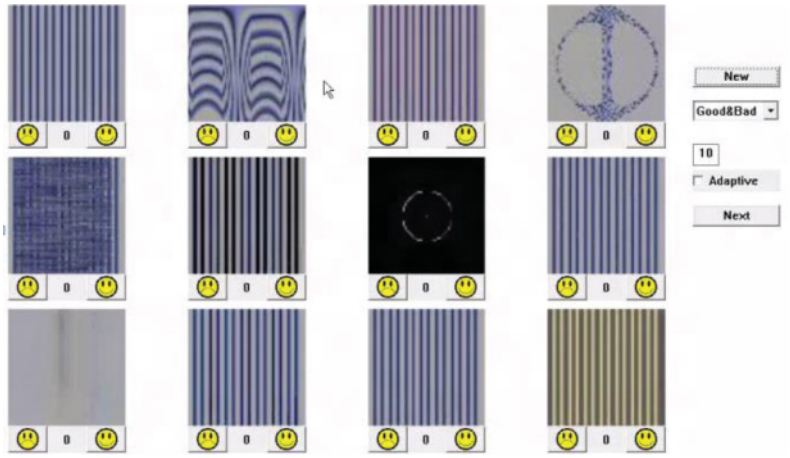


Fig. 10. Image retouching interface.

Table III. Structure of a Gene

Blur	gblur, mblur, pixelize
Enhance	antialias, deinterlace, destripe, nlfilter, red-eye-removal, sharpen, unsharp-mask
Distorts	dog, edge, laplace, neon, sobel, dilate, erode, apply-canvas, cartoon, cubism, oilify, photocopy, soft-glow, blinds, emboss, lens-distortion, polar-coords, ripple, shift, vpropagate, video, waves, whirlpitch, wind
Etc	color-balance, levels, invert, flip, copy-layer, loop

Filters have their parameters to adjust the results. The chromosome also contains information on the details of parameters for each filter. The number of parameters is different for each filter type. However, it ranges from 3 to 5. It should have the order of filters and their parameters. The chromosome is automatically converted into a script-fu for GIMP. Because the number of filters cannot be determined a priori, our evolutionary system uses a variable-length chromosome. It starts from an empty chromosome, but it gradually increases the length from mutation operations.

For each filter, 4 bytes are assigned to represent parameters and filter type. In total, 40 image filters are used (Table III), such as color balance, copy layer, loop, and so on. However, we exclude several filters and complex functions, such as a 3×3 mask, because errors occur. The first byte is used to represent the type of filter (0–39). The other 3 bytes are used to represent parameters of the filter. If the number of filters for a chromosome is N, the size is $4 \times N$ bytes.

If the number of parameters for the filter is three, each parameter is represented with 1 byte. However, some filters have four or five parameters. In this case, 1 or 2 bytes are divided into smaller parts to represent more than two parameters (Figure 11).

Unlike the 3D model chromosome, the texture evolution uses variable-length chromosomes. In the initialization, each chromosome assigns one filter randomly chosen. Because the mutation operator allows for the addition and deletion of filters in the chromosome, the length of the filter sets can be changed. It uses a 1-point crossover and mutation operators.

5.3. Interface

The communication between the GIMP and the IGA interface is done using an I/O file. Each chromosome is converted into a script-fu file, and the GIMP outputs one result

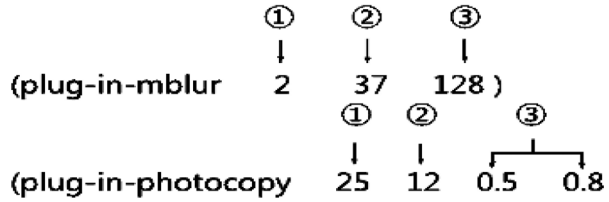


Fig. 11. An example of chromosome representation. (For plug-in photocopy,the last byte is divided into 4 bits to represent two parameters in a byte).

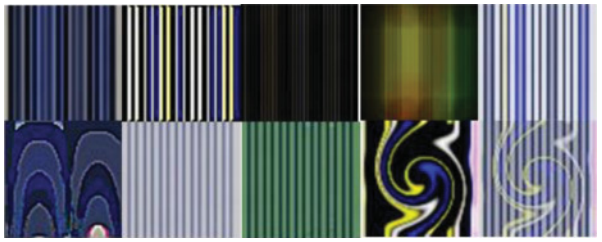


Fig. 12. Some example images from GIMP batch processing.

image file. In the interface, users select their own photos or patterns to create new textures for the 3D model. The evaluation by the user can be converted to the numeric value. For example, the five star input can be interpreted as 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0. The slide scale represents a real value from 0 to 1.

Figure 12 shows an example images created from the IGA evolution. It starts from an original image and a set of filters can transform it to a variety of novel textures.

6. APPLYING TO A REAL GAME

In this section, we show that the new contents can be used in racing games (TORCS, The open Racing Car Simulator). Because this game uses a 3D model format (AC3D) and track toolkit, it is an easily editable game. The web-interface can export the 3D building model in AC3D format. Users just simply download the converted file and insert it into the game folder. The texture evolved was used for the buildings. Because the TORCS only accepts RGB format, we converted the image with GIMP. Figure 13 shows the final racing screenshots with the evolved buildings and textures.

7. USER TEST

In our study, we tested about 3D model generation and texture retouch method. Each test was run by an amateur who does not know to design 3D model. The first test is 3D model generation. The test was performed by seven users using a random based algorithm and our method (IGA). In this case, variable-length IGA gets a higher evaluation than the L-system–based IGA. Therefore, we describe former results. In addition, the image retouching test was run to six users. Table IV shows the results of the survey. Each user evaluate personal satisfy about result contents. The results show that 3D model generation is a useful method for amateur users. Unfortunately, the evaluation results for image retouching were similar to that of random, but this does not mean it is useless. It can be improved the performance as more research.

8. CONCLUSIONS AND FUTURE WORKS

In this article, we propose a 3D model and texture generating method that enables an amateur to make new content for games using IGA. In this work, we still have many

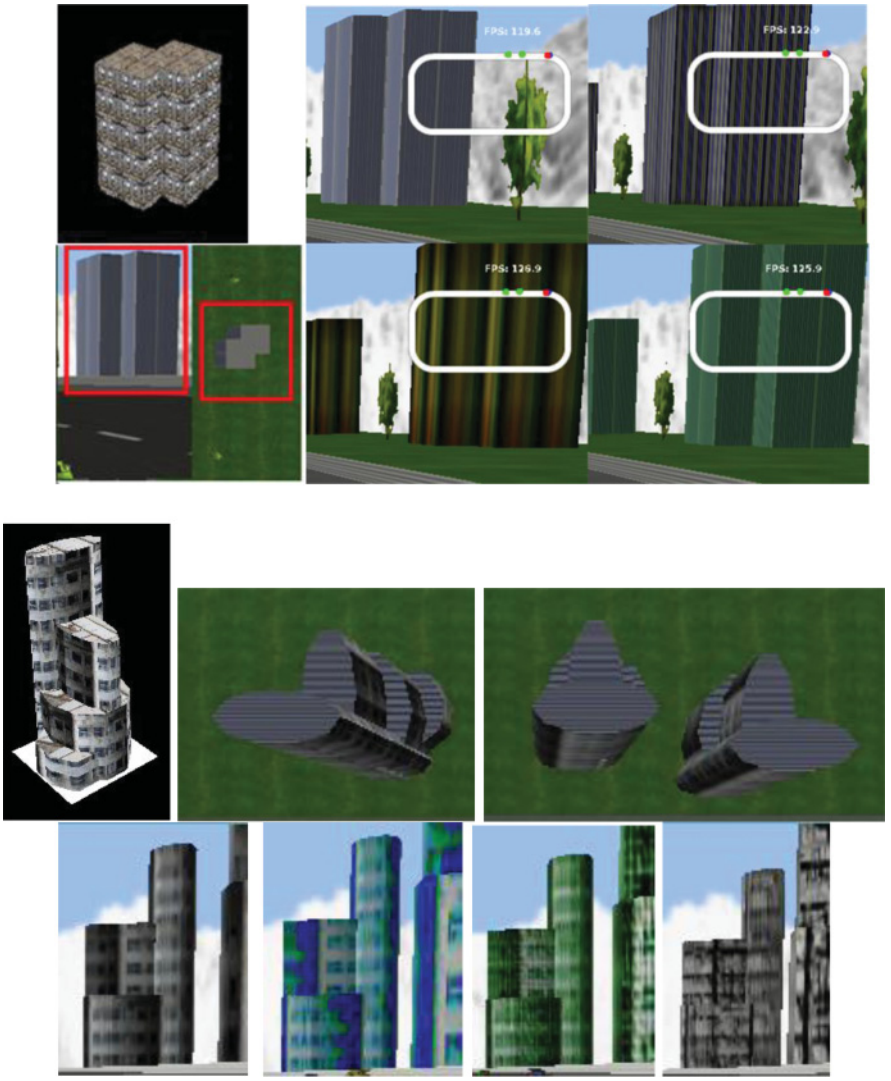


Fig. 13. Running TORCS games with the created contents.

Table IV. Test Results

	Method	Min	25%	Median	75%	Max
3D Model Generation	IGA	4	5	8	8	8.5
	Random	3	5	5	7	7
Image Retouching	IGA	6	7	8	8	10
	Random	5	6	8	9	9

constraints (e.g., fixed initial building footprint size) to generate complex 3D building models. But it shows that the IGA is promising tool to produce 3D models automatically from the feedback of users. The new contents generation system allows users to create their own game content and share them with others.

We plan to generate more complex 3D buildings and extend our system to other types of game objects (cars, characters, etc.). In addition, we need to integrate the structure and texture evolution into a single system.

REFERENCES

Barbaros Bostan and Ugur.Kaplancali. 2010. Explorations in player motivations: Game mods. In *Proceedings of GAMEON-ASIA 2010 Conference*.

Benjamin Bucior and Summer Ventures. 2007. Using Script-Fu in the GNU Image Manipulation Program to Automate "Smart" Sharpening. Summer Ventures, Appalachian State University. DOI: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.8540

Luigi Cardamone Politecnico, Daniele Loiacono Politecnico, and Pier Luca Lanzi. 2011. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)*. ACM, New York, NY, 395–402. DOI: http://dx.doi.org/10.1145/2001576.2001631

Jeff Clune and Hod Lipson. 2011. Evolving 3D objects with a generative encoding inspired by developmental biology. *ACM SIGEVOlution* 5, 4, 2–12. DOI: http://dx.doi.org/10.1145/2078245.2078246

Jonathan Eisenmann, Matthew Lewis, and Bryan Cline. 2011. Interactive evolution for designing motion variants. In *Computational Intelligence.* Springer, Berlin, 135–149. DOI: http://link.springer.com/chapter/10.1007/978-3-642-20206-3_9

Kenji Hara, Atsuhiko Maeda, Hirohito Inagaki, Minoru Kobayashi, and Masanobu Abe. 2009. Preferred color reproduction based on personal histogram transformation. *IEEE Transactions on Consumer Electronics*. 55, 2, 855–863. DOI: http://dx.doi.org/10.1109/TCE.2009.5174466

Erin J. Hastings and Kenneth O. Stanley. 2010. Interactive genetic engineering of evolved video game content. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (PCGames'10)*. ACM, New York, NY, Article 8. DOI: http://dx.doi.org/10.1145/1814256.1814264

Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. 2009. Interactive evolution of particle systems for computer graphics and animation. *IEEE Transactions on Evolutionary Computation* 13, 2, 418–432. DOI: http://dx.doi.org/10.1109/TEVC.2008.2004261

Rudolf Jaksa, Shota Nakano, and Hideyuki Takagi. 2003. Image filter design with interactive evolutionary computation. In *Proceedings of the IEEE International Conference on Computational Cybernetics (ICCC'03)*. 175.

Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. 2011. Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3, 245–259. DOI: http://dx.doi.org/10.1109/TCIAIG.2011.2163692

George Marsaglia. 2003. Random number generators. *Journal of Modern Applied Statistical Methods*.

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. In *Proceedings of ACM SIGGRAPH 2006 Papers (SIGGRAPH'06)*. ACM, New York, 614–623. DOI: http://dx.doi.org/10.1145/1141911.1141931

Dana Petcu and Victoria Iordan. 2006. Grid service based on GIMP for processing remote sensing images. In *Proceedings of 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*. 251–258. DOI: http://dx.doi.org/10.1109/SYNASC.2006.40

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 2012. *The Algorithmic Beauty of Plants*. Springer Science & Business Media.

Jimmy Secretan, Nicholas Beato, David B. D. Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. 2008. Picbreeder: Evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08).* ACM, New York, NY, 1759–1768. DOI: http://dx.doi.org/10.1145/1357054.1357328

Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. 2011. What is procedural content generation? Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games (PCGames'11).* ACM, New York, NY, Article 3. DOI: http://dx.doi.org/10.1145/2000919.2000922

Crystal Valente and Reinhard Klette. 2010. Artistic emulation - filter blending for painterly rendering. In *Proceedings of 4th Pacific-Rim Symposium on Image and Video Technology (PSIVT)*. 462–467. DOI: http://dx.doi.org/10.1109/PSIVT.2010.84

Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant architecture. In *Proceedings of ACM SIGGRAPH 2003 Papers (SIGGRAPH'03)*. ACM, New York, NY, 669–677. DOI: http://dx.doi.org/10.1145/882262.882324

Du-Mim Yoon and Kyung-Joong Kim. 2012. Comparison of scoring methods for interactive evolutionary computation based image retouching system. In *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion Genetic and Evolutionary Computation Conference (GECCO Companion'12)*. ACM, New York, NY, 617–618. DOI:http://dx.doi.org/10.1145/2330784.2330887

Received March 2013; revised May 2013; accepted September 2013