

Generation of Cartoon 2D Cracks Based on Leaf Venation Patterns

Julia M. Colleoni Couto, Tiago Marcelino, Henry Braun, Diogo Strube de Lima, Soraia Raupp Musse
 Graduate Programme in Computer Science
 PUCRS - Av. Ipiranga, 6681, Porto Alegre, RS, Brazil

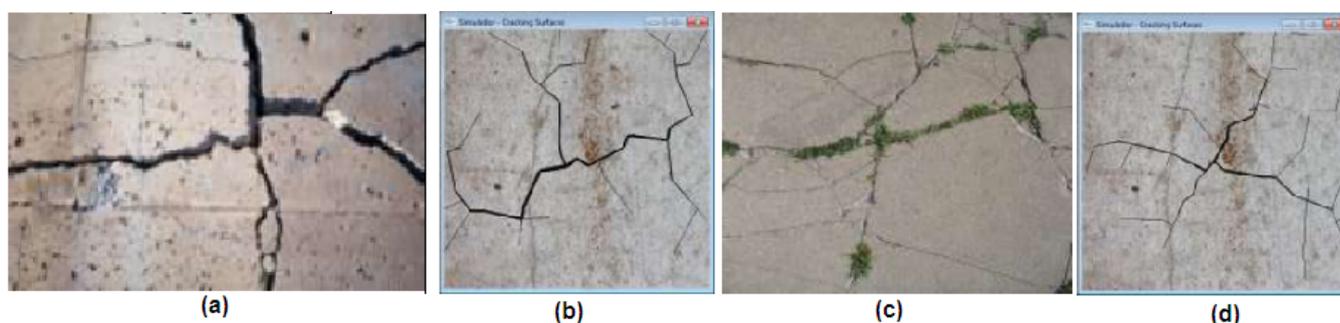


Figure 1: In (a) and (c) real images of cracks in concrete are presented. In (b) and (d), results obtained with our model based on Leaf venation method are illustrated.

Abstract

This paper presents a model for generating 2D cracks in real-time for application in games and cartoon animations. Cracks are generated based on a method previously used for leaf venation patterns. We describe a procedural model to generate cracks that can be customized for application in games, generating different visual results depending on interactive parameters. Results were visually evaluated by users and indicate that hypothesis to adopt leaf venation patterns to generate cracks is valid and possible for interactive applications.

Keywords:: Leaf venation, Crack patterns.

Author's Contact:

juliacolleoni@gmail.com,
 aelle.king@gmail.com,
 henry.braun@acad.pucrs.br,
 diogo.lima@acad.pucrs.br,
 soraia.musse@pucrs.br,

1 Introduction

Simulation of cracks and fractures in real-time are still a challenge. It is due to the fact that Physical simulations require computational power, and when included in a game development, it can compete with many others technical processing. Some work are related in literature describing techniques to provide realistic simulations of 3D and surface cracks and fractures, however none of them have been integrated in real-time applications, as discussed in Section 2.

This paper presents a model to provide 2D cracks in real-time, focused on games and cartoons animations, based on leaf venation patterns. The used model was proposed by Runions et al. [Runions et al. 2005]. Our hypothesis is that same procedural technique used to model leaf venations can be used to generate of some type of cracks. So, we used and adapted such model, as well as evaluated obtained results.

This paper is organized as follows: in Section 2 we present some methods described in literature to generate cracks and fractures. Section 3 describes succinctly the original model proposed by Runions et al. [Runions et al. 2005], while in Section 4 we describe our adaptations to generate cartoon 2D cracks in real-time. Obtained results are presented in Section 5, while final discussions and future work are presented in Section 6.

2 Related Work

There are some methods proposed in computer graphics domain which aim to generate crack surfaces, and they are often grouped into two categories. One of them follows a non-physical approach for generating crack patterns, such as mapping crack patterns to a surface. Other category uses physically based methods to generate cracks. Our work is focused on first category, since we use a space colonization algorithm to consider regions where to generate the 2D cracks.

Some work in literature present techniques for modeling fracture in solid materials, using physically based simulation. Terzopoulos et al. introduced simulating elastic [Terzopoulos et al. 1987] and inelastic deformation of objects, including fractures generation [Terzopoulos and Fleisher 1988]. A mass-spring [Norton et al. 1991] and finite elements [O'Brien and Hodgins 1999] techniques were later used to generate fractures. Federl and Prusinkiewicz used finite elements to model cracks formed by drying mud and tree bark [Federl and Prusinkiewicz 2002],[Federl and Prusinkiewicz 2004].

There are some non-physical approaches to generating surface cracks. One type of approach uses an input pattern or image to generate surface cracks. Is the case of methods proposed by [Martinet et al. 2004], [Desbenoit et al. 2005] and [Mould 2005].

In contrast, our method creates cracks based on space colonization algorithm proposed by [Runions et al. 2005]. We adapted such algorithm in order to provide visually acceptable results in 2D cracks, for animation and generation in real-time, focusing on games application and cartoon animation.

3 The Model to Create Leaf Venation Patterns

The method proposed in this paper is based on the space colonization algorithm, which was originally proposed to model leaf venation patterns [Runions et al. 2005].

The venation model simulates three processes within an iterative loop: leaf blade growth, the placement of markers in the free space, and the addition of new veins. The markers correspond to sources of the plant hormone auxin, which, according to a biological hypothesis [Aloni et al. 2003], emerge in the growing leaf regions not penetrated by veins. A set of auxins S interacts with the vein pattern, which consists of a set of points V called *vein nodes*. This pattern is extended iteratively toward the auxins of free space. The auxins that are approached by the advancing veins are gradually removed, since the space around them is no longer free. As the leaf

grows, additional auxins of free space are added in the space between existing veins and other auxins. This process continues until the growth stops, and there are no auxins left.

The interplay between auxins of free space and vein nodes is at the heart of the space colonization algorithm. During each iteration, a vein node is influenced by all the auxins closer to it than any other vein node. Thus, veins compete for auxins, and thus space, as they grow. There may be several auxins that influence a single vein node v : this set of points is denoted by $S(v)$. If $S(v)$ is not empty, a new vein node v' will be created and attached to v by an edge representing a vein segment. The node v' is positioned at a distance D from v , in the direction defined as the average of the normalized vectors toward all the auxins $s \in S(v)$. Thus, $v' = v + D\hat{n}$, where

$$\hat{n} = \frac{\mathbf{n}}{\|\mathbf{n}\|} \quad \text{and} \quad \mathbf{n} = \sum_{s \in S(v)} \frac{s - v}{\|s - v\|}. \quad (1)$$

The distance D serves as the basic unit of distance in the model and provides control over the resolution of the resulting structure. Once the new nodes have been added to V , a check is performed to test which, if any, of the auxins of free space should be removed due to the proximity of veins that have grown toward these points.

The space colonization algorithm has subsequently been adapted to model trees [Runions et al. 2007]. Beyond the extension to 3D structures, the algorithm for trees introduced the notion of the radius of influence, which limits the distance from which auxins of free space can attract tree nodes. Furthermore, the set of auxin points is usually predefined at the beginning of simulation and no new auxins are added afterwards, since, in contrast to the expanding leaf blade, the space in which a tree grows remains fixed.

Next section describes our model to create 2D cracks based on venation patterns for interactive applications.

4 The Model to Create 2D Cracks

This section describes all required adaptations made in original model in order to generate cartoon 2D cracks. Firstly, we discuss the method used for colonize the space, then we present the main characteristics of our model as well as steps used to customize cracks generation.

4.1 The Colonization Algorithm

Runions et al [Runions et al. 2005] proposed the generation of auxins based on Dart Throwing algorithm. Auxin sources emerge at locations that are farther than a threshold birth distance from the other sources, and farther than a threshold birth distance bv from the vein nodes. In original model, authors compute these points using a version of the dart-throwing algorithm [Cook 1988], [Mitchell 1987], which controls the regularity of the venation patterns. The drawback is the computational cost of dart-throwing algorithm, since it consists of repeatedly generating points distributed at random, then testing each new point against all points already in the set. A point that is sufficiently far from other points is accepted as a new member of the set.

In our case, we adopted a pseudo-regular grid in order to generate the markers (as we called instead of auxins) used to populate the space. The markers are randomly placed in the vertex of a grid, whose number of cells are defined by the user. The randomness process, i.e. to generate or not an auxin on a vertex, is used in order to provide aleatory cracks, and should keep a pre-defined percentage of space colonized. Figure 2 illustrates the markers and shows a growing crack. It is important to emphasize that it is not the first time the Runions model is used for other goals. It has been used to simulate the motion of crowds of virtual humans [Rodrigues et al. 2009].

Yet, one can change the number of lines and columns existent in the grid. Figure 3 shows two images using different number of lines and columns. On the left, used resolution is 20x20, while on the right is 100x100.

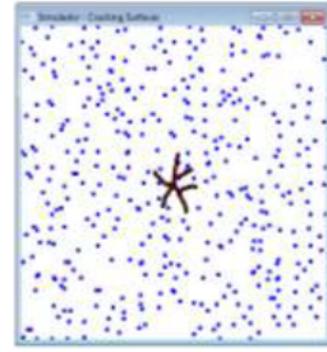


Figure 2: Image illustrates the created markers using pseudo-regular grid and beginning of a crack generation.

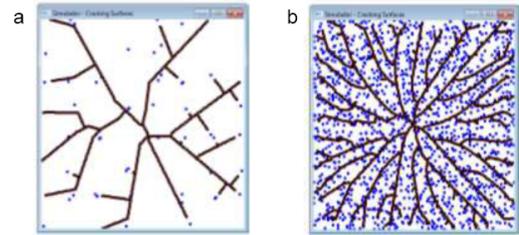


Figure 3: Two images illustrating different resolution of lines and columns in the grid used to colonize space with markers. On the left the resolution is 20x20, while on the right is 100x100.

4.2 Cracks Generation

The crack starts its veins from an initial point. For simplification, in this paper, we consider only a fixed start point in the center of the image, but we can easily change configuration in order to provide cracks growing from any initial location. It could be the application of a shooter game, for instance. Moreover, we can develop cracks starting at more than one location simultaneously. Consequently, cracks generated in glass surfaces could present better visual results, if for example all starter points could be located in the region defined in the center of the crack (see Figure 11).

Once the initial position is defined, the crack starts to evolve. Based on the computation proposed by Runions et al [Runions et al. 2005] and described in Section 3, new crack nodes (in original model they are called vein nodes) are created, and lines are drawn between them.

Firstly, the algorithm should find out which markers (m) are close to each crack node c . This set of markers is denoted by $S(c)$. If S is not empty, a new crack node c' is created and attached to c . Afterward, lines are drawn between c and c' . The node c' is positioned at a predefined distance D from c , in the direction defined as the average of the normalized vectors toward all the markers m close to c , $c' = c + D\hat{n}$, where

$$\hat{n} = \frac{\mathbf{n}}{\|\mathbf{n}\|} \quad \text{and} \quad \mathbf{n} = \sum_{m \in S(c)} \frac{m - c}{\|m - c\|}, \quad (2)$$

similarly to Equation 1.

Following parameters are the full list considered in our model:

1. *kill_distance*: responsible for defining the threshold distance from crack node to markers that should be removed (default value is equal to 0.01). Figure 4 illustrates two images generated using two different values for *kill_distance* parameter. In a) *kill_distance* is equal to 0.01, while in b) is 0.1. It is possible to see that image a) generates more crack nodes than in b). Other parameters used in these simulations are: $D = 0.2$, $crack_radius = 0.9$, $num_iterations = 100$ and $num_lines_grid = num_column_grid = 7$.

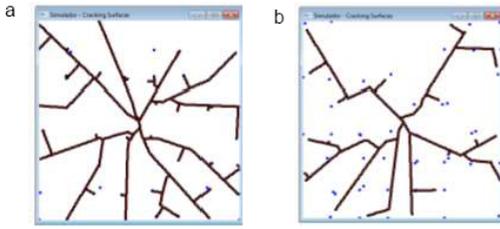


Figure 4: Two images illustrating different values for *kill_distance* parameter. In a) *kill_distance* is equal to 0.01, while in b) it is 0.1. It is possible to see that image a) generates more crack nodes than in b).

2. *D*: distance from a crack node to create another node (default value is equal to 0.1). Depending on the used value, crack segment generated between two crack nodes are smaller, for small *D*, or larger for large *D*. In Figure 5(a), the default value is used, while in (b) *D* is equal to 0.04. It is possible to see that image a) generates less crack nodes than in b). Other parameters used in these images are: *kill_distance* = 0.08, *crack_radius* = 0.9, *num_iterations* = 100 and *num_lines_grid* = *num_column_grid* = 20.

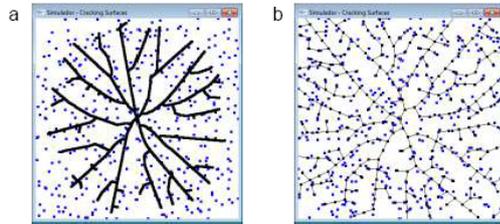


Figure 5: Two images illustrating different values for *D* distance parameter. In a) *D* is equal to 0.1, while in b) it is 0.04. It is possible to see that image a) generates less crack nodes than in b).

3. *num_interaction*: this parameter describes number of interactions executed during the simulation (default value is equal to 100).
4. *num_lines_grid* and *num_columns_grid*: describe the number of lines and columns included in the pseudo-grid used to generate markers (default value is equal to 20). We called it as pseudo-grid since not all vertex in the grid generates markers, due to the fact that is randomly decided. Figure 3 illustrates two images containing different resolutions in the grid.
5. *crack_radius*: is the space around the crack nodes where markers are evaluated to find out the close nodes to be associated, and consequently generate new crack node (default value is equal to 0.9). This parameter has been proposed in original model and maintained in our method. As the value of *crack_radius* is decreased, more instability exists among markers which should influence the generation of new crack node. This influence is visually represented as the curvature present in final produced cracks. In Figure 6 it is possible to observe this characteristic when comparing images (a) and (b), where *crack_radius* used are 0.5 and 20.0, respectively. Other parameters used in these simulations are: *kill_distance* = 0.1, *D* = 0.2, *num_iterations* = 100 and *num_lines_grid* = *num_column_grid* = 20.
6. *line_growth*: describes the line width growing as a function of hierarchy of crack nodes (default value is equal to 0). This parameter was included to improve the realism of generated cracks, specifically in clay and concrete surfaces. If *line_growth* is different to 0, after a crack node *c* generates all possible *c'*, the line width (*lw*) used to draw the crack segments is increased by a growing factor computed using *line_growth* and the number of generated crack nodes *c'*: $lw = line_growth \cdot n \cdot c'$, where $n \cdot c'$ is related with the number of generated *c'*. Figure 7 shows two images illus-

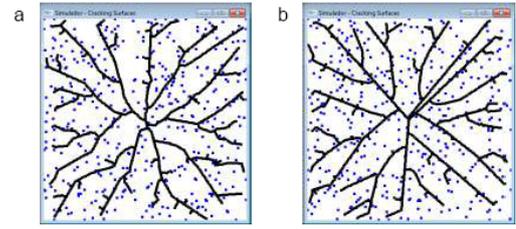


Figure 6: Two images illustrating different values for *crack_radius* parameter. In a) *crack_radius* is equal to 0.5, while in b) it is 20.0. It is possible to see that image b) generates straighter crack segments than in a).

trating the effect of *line_growth* parameter. In (b) we can observe the variation in line width generated as a function of hierarchy process of cracks generation. Other parameters used in these simulations are: *kill_distance* = 0.08, *D* = 0.2, *num_iterations* = 100 and *num_lines_grid* = *num_column_grid* = 20.

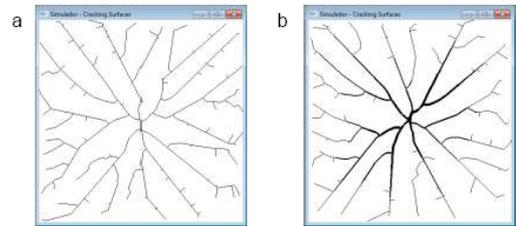


Figure 7: Two images illustrating different values for *line_growth* parameter. In a) *line_growth* is equal to 0.5, while in b) it is 0.0. It is possible to see that image b) generates varied lines width as a function of cracks hierarchy in generation process.

7. Visualization parameters are related with visualization aspects of the generated cracks. Such parameters are represented with boolean data (TRUE/FALSE). They are: *draw_markers* (an example is illustrated in Figure 2), used to visualize the markers in the space; *draw_connections* allows the line draw among crack nodes *c'* generated from same *c*. This parameter was proposed in order to improve the visual quality of generated cracks in glass surfaces. Yet, *background_image* is related with an image used in the background of the crack. Figures 8 and 9 show images comparing the effect of these two last parameters.

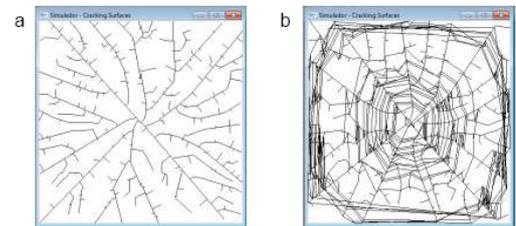


Figure 8: Two images illustrating the effect of *draw_connections* parameter. In a) *draw_connections* is FALSE, while in b) it is TRUE. Other parameters used in these simulations are: *kill_distance* = 0.02, *D* = 0.2, *num_iterations* = 80 and *num_lines_grid* = *num_column_grid* = 20.

A last visualization parameter *draw_islands* is used to generate specific pattern in the cracks, by creating connections among close crack nodes. The difference between *draw_connections* and *draw_islands* is that the first one draws connections among crack nodes generated from the same crack node, while the last one connects crack nodes close to each other. Figure 10 shows the impact of *draw_islands* parameter.

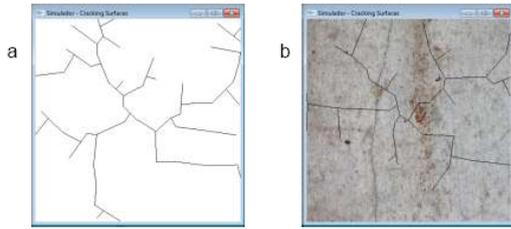


Figure 9: Two images illustrating the effect of *background_image* parameter. In a) *background_image* is *FALSE*, while in b) it is *TRUE* (so, it is possible to chose the background image file). Other parameters used in these simulations are: *kill_distance* = 0.08, $D = 1.0$, *num_iterations* = 100 and *num_lines_grid* = *num_column_grid* = 7.

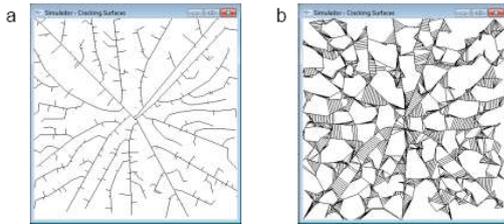


Figure 10: Two images illustrating the effect of *draw_islands* parameter. In a) *draw_islands* is *FALSE*, while in b) it is *TRUE*. Other parameters used in these simulations are: *kill_distance* = 0.02, $D = 0.2$, *num_iterations* = 80 and *num_lines_grid* = *num_column_grid* = 20.

Next section discusses obtained results. We conducted a survey with subjects in order to evaluate the results of our method. Considerations about such evaluation is also discussed in this section.

5 Results

In this section we show some results obtained by using our prototype. Furthermore, results have been evaluated by 84 subjects in a survey format. Results of such evaluation are also presented in this section. It is important to say that our model does not aims to present realistic results but cracks for cartoon and games.

We specified our results in three type of materials: clay, glass and concrete. Furthermore, we offer a visual comparison with real images from real life, available on the internet. For first simulations, trying to simulate glass surfaces, we used following parameters:

- *kill_distance* = 0.02;
- $D = 0.8$;
- *num_iterations* = 80;
- *crack_radius* = 1.0;
- *num_lines_grid* = *num_column_grid* = 20;
- *draw_markers* = *FALSE*;
- *background_image* = *glass.bmp*;
- *draw_connections* = *FALSE* for black lines and *TRUE* for white lines;
- *line_growth* = 0 for black lines and 0.5 for white lines; and
- *draw_islands* = *FALSE*.

Figures 11 and 12 show results of crack generation for glass surfaces, as well as real cracks, captured from real scenes.

We also generate cracks trying to simulate the effect of concrete surfaces by specifying following values for our parameters:

- *kill_distance* = 0.08;

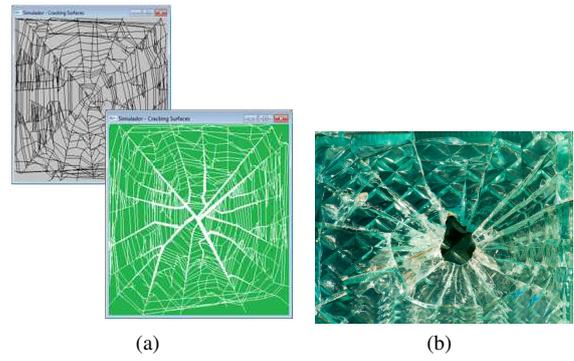


Figure 11: (a) Images generated by using our model; (b) Real effect of cracks in glass surfaces.

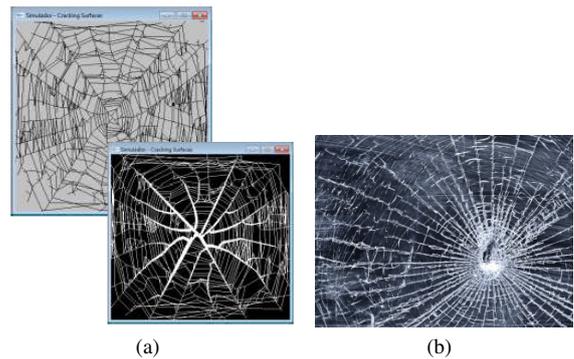


Figure 12: ((a) Image generated by using our model; (b) Real effect of cracks in glass surfaces.

- $D = 1.0$;
- *num_iterations* = 100;
- *crack_radius* = 0.4;
- *num_lines_grid* = *num_column_grid* = 7;
- *draw_markers* = *FALSE*;
- *background_image* = *concrete.bmp*;
- *draw_connections* = *FALSE*;
- *line_growth* = 1; and
- *draw_islands* = *FALSE*.

Figures 13 and 14 show results of crack generation for concrete surfaces.

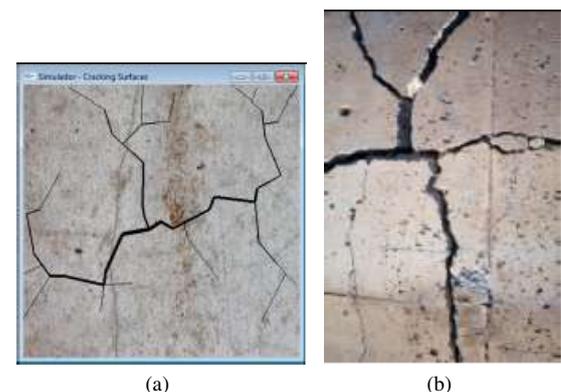


Figure 13: (a) Image generated by using our model; (b) Real effect of cracks in concrete surfaces.

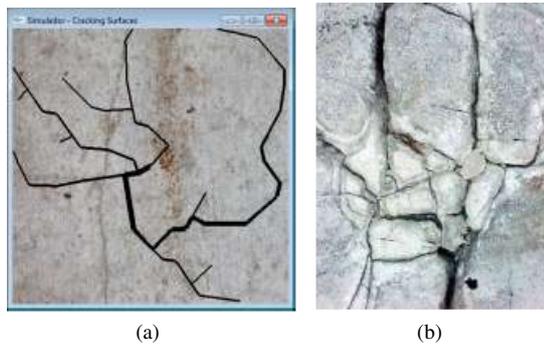


Figure 14: (a) Image generated by using our model; (b) Real effect of cracks in concrete surfaces.

Lastly, we changed parameters in order to simulate cracks in clay surfaces. From the three types of simulated surfaces, clay cracks present more difficulties since in real life it generates very visible clusters of surfaces, when it cracks. That was the main motivation for us to propose the *draw_islands* parameter. However, we believe better results could be achieved in a future work, as discussed in next Section. Simulation of cracks in clay surfaces is certainly a limitation of the current version of our model. In images illustrated in following Figures, we used following parameters:

- *kill_distance* = 0.08;
- *D* = 0.8;
- *num_iterations* = 100;
- *crack_radius* = 0.4;
- *num_lines_grid* = *num_column_grid* = 20;
- *draw_markers* = *FALSE*;
- *background_image* = *brown.bmp, gray.bmp*;
- *draw_connections* = *FALSE*; and
- *line_growth* = 0.5.

Figures 15, and 16 show results of crack generation for clay surfaces. In Figure 15 the parameter *draw_islands* was set to *FALSE*, while in Figure 16 it was set to *TRUE*.

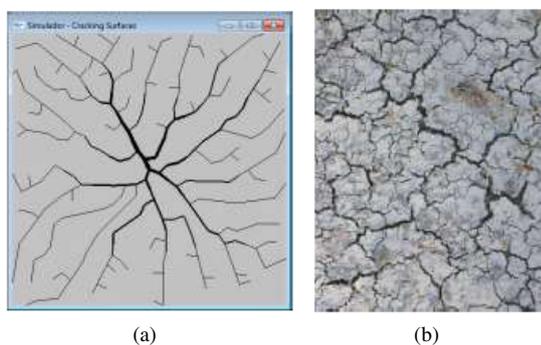


Figure 15: (a) Image generated by using our model; (b) Real effect of cracks in clay surfaces.

All the simulations (from the beginning until the final image generation) are processed in real time (more than 30 FPS). More specifically, the computational time is dependent of the number of considered markers. Low frame rates were achieved when simulating glass surfaces (30 FPS), while high frame rates were achieved in concrete surfaces (more than 100 FPS). The crack generation can be performed to generate also an animation, as illustrated in 3 frames from 100 processed, as showed in Figure 17.

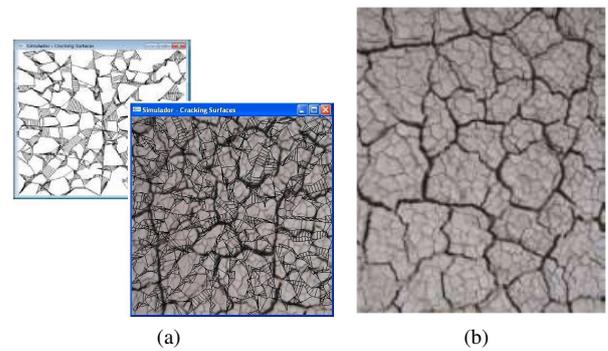


Figure 16: (a) Image generated by using our model including *draw_islands* process; (b) Real effect of cracks in clay surfaces.

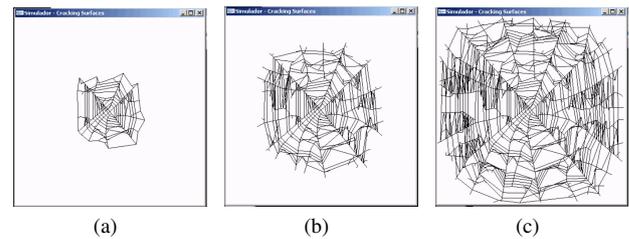


Figure 17: Frames of animation to generate crack in glass surfaces.

5.1 Evaluation

In order to visually evaluate our results, we conduct a survey where approximately 84 subjects observe images generated in our model, and answered 3 simple questions, regarding the three surfaces we worked on (concrete, glass and clay). Subjects should answer if images simulating cracks were a) totally similar, b) similar, c) marginal d) different and e) totally different, which weights for answers were included in interval [5; 1], respectively.

Figure 18 shows the results obtained with experiments including approximately 90 subjects opinions. We can observe that best evaluated scenario occurred when we simulated concrete surfaces, followed by glass. The simulation of cracks in clay surfaces obtained low scores in subjects evaluation, as expected, mainly due to the fact that clusters and separated polygons can not be observed, when using our model. Even if we set *draw_islands* as *TRUE*. This is a main current limitation of our model and should be improved in next work.

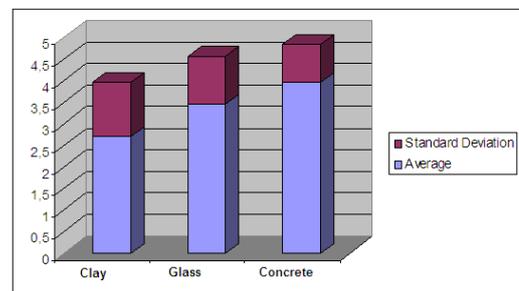


Figure 18: Result of evaluation obtained in survey answered by approximately 90 subjects regarding 3 simple questions about the visual quality of cracks simulating e different surfaces: clay, glass and concrete.

6 Final Considerations

This paper presented a model for cracks generation based on space colonization algorithm, originally used for leaf venation modeling. Our model is focused on providing crack generation for cartoon and

games, since the model is not Physically coherent. We propose a model based on few parameters: 5 parameters for geometric crack generation and 5 for visualization purposes. These parameters have been investigated and varied in order to investigate the crack simulation in three specific surfaces: clay, glass and concrete. All results are achieved in real time, and provide generation of crack image, as well the animation of the hole process.

In addition, our model has been qualitatively evaluated. Approximately 90 subjects evaluated obtained results in terms of visual quality. This analysis showed that clay surfaces obtained low score of subjects answers and more work should be done in this scenario. Future work involves improve the crack generation in clay surfaces, and also provide the subdivision of geometry, by generation polygons and consequently fractures of surfaces.

References

- ALONI, R., SCHWALM, K., LANGHANS, M., AND ULLRICH, C. 2003. Gradual shifts in sites of free auxin-production during leaf-primordium development and their role in vascular differentiation and leaf morphogenesis in *Arabidopsis*. *Planta* 216, 841–853.
- COOK, R. L. 1988. Stochastic sampling in computer graphics. 283–304.
- DESBENOIT, B., GALIN, E., AND AKKOUICHE, S. 2005. Modeling cracks and fractures. *The Visual Computer* 21, 8-10, 717–726.
- FEDERL, P., AND PRUSINKIEWICZ, P. 2002. Modelling fracture formation in bi-layered materials, with applications to tree bark and drying mud. In *Proc. of the 13th Western Computer Graphics Symp.*
- FEDERL, P., AND PRUSINKIEWICZ, P. 2004. Finite element model of fracture formation on growing surfaces. In *Lecture Notes in Computer Science 3037 (Jan 2004)*, 138145.
- MARTINET, A., GALIN, E., DESBENOIT, B., AND AKKOUICHE, S. 2004. Procedural modeling of cracks and fractures. *Shape Modeling and Applications, International Conference on 0*, 346–349.
- MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph.* 21, 4, 65–72.
- MOULD, D. 2005. Image-guided fracture. In *GI '05: Proceedings of Graphics Interface 2005*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 219–226.
- NORTON, A., TURK, G., BACON, B., GERTH, J., AND SWEENEY, P. 1991. Animation of fracture by physical modeling. *Vis. Comput.* 7, 4, 210–219.
- O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 137–146.
- RODRIGUES, R., BICHO, A., PARAVISI, M., JUNG, C. R., MAGALHES, L. P., AND MUSSE, S. R. 2009. A model for generating and animating groups of virtual agents. *Lecture Notes in Computer Science 5773*, 358–371. In: Proceedings of 9th International Conference on Intelligent Virtual Agents.
- RUNIONS, A., FUHRER, M., LANE, B., FEDERL, P., ROLLAND-LAGAN, A., AND PRUSINKIEWICZ, P. 2005. Modeling and visualization of leaf venation patterns. *ACM Transactions on Graphics* 24, 3, 702–711.
- RUNIONS, A., LANE, B., AND PRUSINKIEWICZ, P. 2007. Modeling trees with a space colonization algorithm. In *Proceedings of the Eurographics Workshop on Natural Phenomena*, 63–70.
- TERZOPOULOS, D., AND FLEISHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proceedings of ACM Computer Graphics-SIGGRAPH 88*, 269278.
- TERZOPOULOS, D., J., P., A., B., AND FLEISHER, K. 1987. Elastically deformable models. In *Proceedings of ACM Computer Graphics-SIGGRAPH87*, 205214.