Realistic Modeling of Self-adapted Spider Orb-webs in Real-time

Wagner Schmitt
Instituto de Informática
UFRGS
Porto Alegre, Brazil
www.inf.ufrgs.br/∼wschmitt

Marcelo Walter
Instituto de Informática
UFRGS
Porto Alegre, Brazil
marcelo.walter@inf.ufrgs.br

João Luiz Dihl Comba
Instituto de Informática
UFRGS
Porto Alegre, Brazil
comba@inf.ufrgs.br

Fig. 1: Three spider orb-webs procedurally generated into a synthetic scene rendered in real-time. After the user specifies a site and a orientation, the web automatically searches for nearby objects where it can attach its threads to form a web structure.

***Abstract*—Modeling organic objects and natural phenomena is a very time consuming task, and usually requires a certain level of expertise. This happens due to the lack of suitable geometric primitives available in most modeling tools. In addition, with the great advance in real-time rendering power, there is a growing demand for more complex models. Hence, this paper proposes a method to automatically build visually convincing spider orb-webs that are self-adapted to the environment. We show results of mesh-based spider orb-webs built in real-time that are visually pleasing and realistically looking.**

***Keywords*-natural phenomena; modeling; rendering; spider webs; orb-webs;**

I. Introduction

Modeling organic objects and natural phenomena can be quite complex. It is usually a task that requires significant expertise and a high amount of time. One of the reasons is that standard geometric primitives available on common modeling programs are not suitable for these purposes [1]. Therefore, in the literature there are many studies on the automatic creation of organic structures such as trees and plants [2], water, and smoke, among others [3].

Spider webs are an example of an interesting construction due to its seemingly complex geometrical structure and its natural beauty. Figure 2 illustrates some examples of real spider webs. The use of such structures in computer generated scenes can add visual complexity and enhance realism, especially in scenes of natural environments. Among all types of webs, the orb-web is the most studied type [4] due to its spiral shape. Biological studies on how spider webs are formed are extensively investigated in [5], [6], [7], and describe how internal and external factors affect the web structure. These studies lead to highly complex biological models that take into account changes and conditions, which in most cases are expensive to simulate computationally.

In this work, we propose an algorithm to build procedurally mesh-based orb-webs that can be displayed in real-time. The input for our algorithm is just the position and orientation of the web and nearby objects where the web threads will be attached. The procedural construction of the web considers some variables that define the properties of the web structure, such as the radii number, or the spiral size, which were

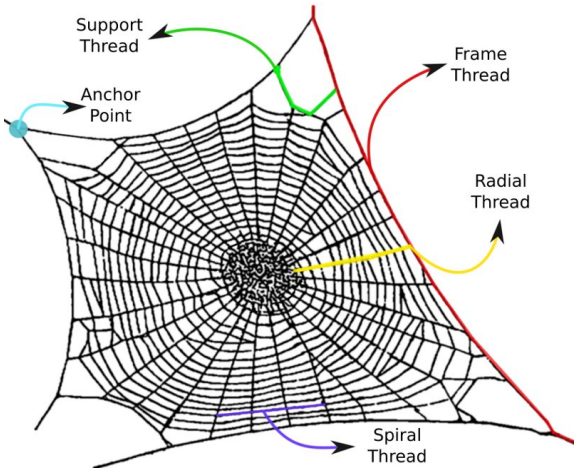


Fig. 3: The names of the web components. Many of these components are created in distinct phases by the spider.

retrieved from biological data [5], [4]. In Figure 1 we illustrate one example of a procedurally generated spider orb-web composed into a synthetic scene.

Contributions: In this paper, we introduce a new method for automatic generation of convincing orb-web models, with a special focus on the geometrical structure and adaptation of the web to the environment. The results are mesh-based spider orb-webs that are visually pleasing and realistically looking, and that can be composed in scenes in a flexible manner. These generated models can enhance the realism of 3D animations and games without much computational cost, especially on the synthesis of natural environments.

A. Related work

Many authors have extensively studied the procedures and techniques that spiders use for modeling their webs [5], [7], [6]. These studies demonstrated that, although the webs appear to be highly complex structures, they are, in fact, constructed in several relatively simple steps. These common build features can be noticed even with spiders of different species [5]. However, there are still no conclusive results on which internal or external factors most affect the construction of the web [4]. Hence, there is still a great difficulty in generating a consistent biological model for creation of orb-webs.

Despite the biological research in the study of construction of orb-webs, there are no approaches in the computer graphics community concerned with a method to procedurally create more convincing models of spider orb-webs. A relevant work for this topic was being developed by Lafortune [8], where the author created a tool to automatically build orb-webs and integrate them into a scene. Unlike ours, his technique uses a ray tracer to achieve high quality non-real-time images. However, his results show spider orb-webs that are much simpler than real ones; the lack of important structures such as support threads reduces the realism of the model.

Another technique was explored by Wiley and Fecho [9], where they attempted to make an orb-web as realistic as possible. However, they were only concerned with the rendering quality, all the modeling part was done manually. They used ray-tracing approaches for rendering, integrated with advanced shader techniques such as refraction, reflection, spectral light dispersion and blooming.

II. Methodology

A. Types of spider orb-webs

Besides the classical spiral orb-webs, which are the focus of this work, there are a few more types of webs that can be found in nature like cobwebs, sheet webs, funnel and tubular webs [10]. Most of them are asymmetrical and present an irregular pattern of threads, making them not visually attractive as the orb-web types, produced by the family of *Orbicularia*. In Figure 2 we show a few examples of real spider orb-webs. Variations occur not only with different types of webs, but even with webs built by spiders of the same species – although, maintaining common important features – since external and internal factors have a great influence in its structure [4].

B. Construction

Spider orb-webs seem to be a simple structure. Superficial analysis usually tend to describe the orb-web as being just some radial lines and a spiral [4]. However, if a more careful analysis is done, it is revealed that the orb-web is a geometrical complex structure consisting of different elements and regions that are built in different phases, as can be seen in Figure 3

To deal with the intricacy of the resulting orb-web structure, the construction algorithm is described in fairly simple geometrical steps, as follows:

1) Anchor points placement;
2) creation of the frame threads;
3) creation of the support threads;
4) populating the orb-web with the radial threads;
5) creation of the spiral.

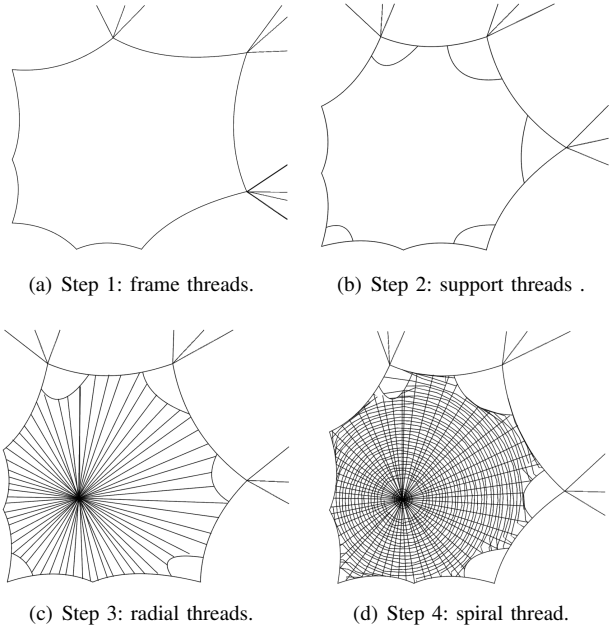
Steps 2 to 5 follow well-known rules, and therefore can be modeled with good precision, based on the data presented in Table I. The building steps are described in Figure 4.

1) Anchor Points Placement: Establishing the anchor points is the first step in the web construction. These points make the connection between the web and the environment, giving the web the necessary stability. In addition, the anchor points define some important characteristics, such as the web total area, and the hub position.

Spiders start the web by making a horizontal line between two fixed points in the environment. The next point is usually on a lower object, such as the ground or lower foliage, which makes the early web structure a fork-shape. After that, the spider searches for more nearby objects to attach its threads.

Our algorithm behaves a little different when choosing the anchor points. First, it must receive a position and a normal orientation from the user. The position is the hub (the center)

Fig. 2: Real close-up pictures of spider orb-webs taken from natural landscapes.



(a) Step 1: frame threads. (b) Step 2: support threads . (c) Step 3: radial threads. (d) Step 4: spiral thread.

Fig. 4: The algorithm building sequence for a orb-web.

of the web – this is the only step that requires any kind of user input. These input values allow us to define a plane in space where the web will be contained. After this, rays are cast from the center position looking out for nearby objects, as the example shows in Figure 5. Whenever a ray successfully finds an object the collision point becomes an anchor point,

TABLE I: Design variability in orb-web models. The values of Radii Number and Spiral Distance Factor, were estimated based on the work of Vollrath [4] and [11], respectively, this values were considering orb-webs that range from small (300cm^2) to normal size (900cm^2). The other values not found in the literature were estimated empirically, concerning only with the visual appearance, not with biological accuracy.

Variables	**Minimum**	**Maximum**
Radii Number	26	35
Anchor Points	3	9
Hub Position Disturbance*	-1.5	1.5
Spiral Distance Factor (b)	0.03	0.04
Spiral Disturbance (a)	0	0.02

Note: * are variations occurring in the x, y axis, relative to the web orientation.

otherwise the ray is discarded. To keep the anchor points between 3 and 9 (Table I), we cast rays separated by angles of about 30 degrees, plus some random variation. If the number of anchor points is greater than 9, the closest ones are neglected. The rays and the plane normal are always perpendicular, which ensures that the web is a 2D structure contained in the plane.

There are two types of anchor points attachment: single or composed. These types are represented in Figure 6. The criteria used to choose the type of an anchor point is based on its distance from the hub: the greater the distance, more weight and tension this anchor point has to support; in this case the best choice is to use a composed anchor point. On the other hand, closest points suffer less tension and do not need more than one point of attachment to stay stable. The distance

limit can vary depending on the scene; it determines the final size of the web. When a ray reaches this limiting distance and does not reach any object, the algorithm casts more rays from where the original ray ended. These new rays travel only a small distance compared with the original ray (about 10%). If exists an object in the environment where these new rays can reach, this anchor point becomes a composed anchor point.

2) Frame Threads: The frame threads connect the previous computed anchor points forming a convex hull. Therefore, this step draws lines between neighboring anchor points, making the web a closed structure. It is important that the anchor points be read in a specific order (the order they were found in the first phase). This avoids the need of computing the convex envelope of the points.

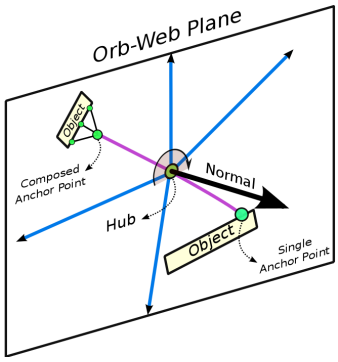
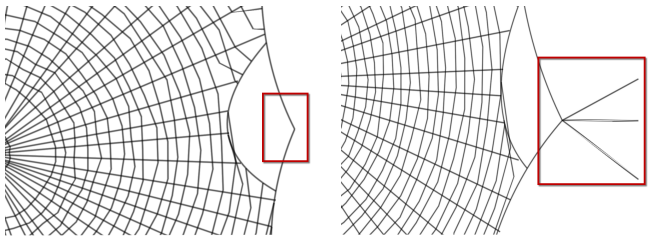


Fig. 5: An example of the anchor point search method used in the first phase of the algorithm. It starts casting rays from the hub that are perpendicular to the plane normal. If the rays collide with an object until a determined distance chosen by the user – this distance will determine the final size of the web, and should be particular for each scene –, this collision point becomes a single anchor point for the web. Otherwise, the anchor points become composed anchor points, connecting more than one thread to the object.



(a) A single anchor point is used when it is fixed not far from the hub, and do not need to support much tension.

(b) A composed anchor point. It attaches to more than one fixed point in the environment. This usually happens when an anchor point is far away from the hub and needs to support a great part of the web tension.

Fig. 6: Red rectangles represent anchor points; these points are fixed and attached to the environment. Their objective is to support part of the tension caused by the weight of the web.

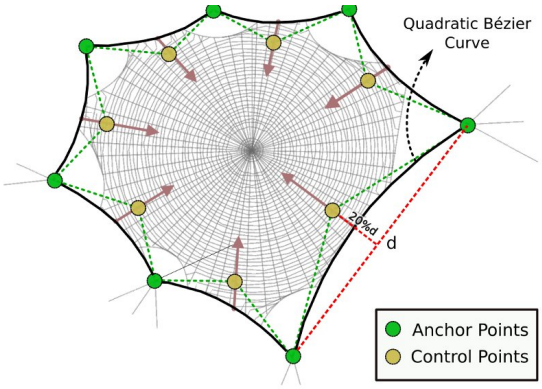
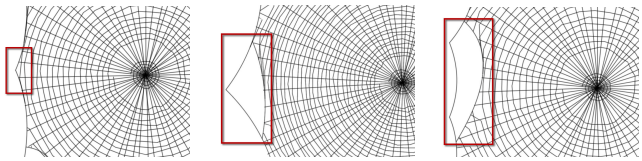


Fig. 7: The frame threads are represented as Bézier curves. Each control point is calculated taking into consideration the length d of its respective frame thread. The amount displaced is 20% of frame thread length towards the directions described by the red arrows.

Another consideration is that the threads connecting two anchor points will suffer a tension by the gravity and the weight of the web structure. This tension usually makes the threads bend a little. It would be very unrealistic if they were rendered simply by using straight lines. In order to simulate the curvatures of the threads we used a quadratic Bézier spline. The control points are displaced towards the hub, by an amount based on the length of the thread (we empirically found that 20% of the length gives a fine curvature, and used this value for all results). The longer the thread is, the more it bends, giving a concavity appearance to the orb-web, as represented in Figure 7.

3) Support Threads: This type of threads, as the name suggests, are primarily used to help supporting the web, ensuring stability. Radial threads will be attached to them afterwards, mitigating the tension that would be handled by the frame threads. To do this efficiently, spiders usually make an arc connecting two neighboring frame treads, as can be seen in Figure 4(b).

The placement of these threads is more complex than the other types, since in real orb-webs this type has a higher



(a) None. (b) Single. (c) Composed.

Fig. 8: Variability of the support threads. If the distance of the anchor point to the hub is too small, case (a) happens. If two anchor points are too close, case (c) happens. Case (b) happens when the anchor point is far away from both the hub and any other neighboring anchor points.

variability, as can be seen in Figure 2, mainly regarding to its initial and final positions. To simulate this type of threads, first we had to decide when and where it is necessary to create a support thread. The more common place is to attach them between neighboring pair of frame threads. Besides that, we assume that if the distance of the common anchor point is too close to the hub, the support thread is not needed, because there is not much tension on this frame thread. If the common anchor point is further away from the hub, a support thread is placed connecting both wires. We also modeled a third case, when we have two neighboring anchor points very close to each other, a very small frame thread is present between the two anchor points, as seen in Figure 8(c). In order to avoid having two support threads very close to each other, which would be very unrealistic, we simply create only one support thread that encompasses both, skipping the small frame thread.

Another important decision was to find in which point of the frame threads the support wires will start and end. By experimentation and observation of real examples, a value that fits well is to start at the 80% of the first frame thread, and end at about 20% of the next, with some small variation of 5%.

Since these threads also need to support some tension, they are drawn with curves using the same strategy as the frame threads, with quadratic Bézier splines. Again, the control points were pulled towards the hub, based on the length of the frame threads.

4) Radial Threads: In this step we place the radial thread segments. They start at the hub and follow a straight line until they connect to another thread, usually a frame thread or a support thread. Radial threads serve as a support for the last phase, the construction of the spiral thread.

The main decision here is the number of radii, which do not follow any strictly rule. The biological factor that most influences the insertion of radii segments is the size of the spider. Hence, we have some freedom deciding this value. In order to find a realistic number of radii threads for the webs, we used the value presented in Table I, that were extracted from statistical data from real orb-webs. The numbers of radii between 26 to 35 give us an angle of 14 to 10 degrees between each radial thread, approximately. This variation is used to mimic the biological variation that happens in real orb-webs.

The radii threads are drawn starting at the hub and stop until they reach a frame thread, a support thread, or an environment obstacle. We defined an iterative process to check when a radial line intersects a frame or support thread. In each iteration, the radial thread starting at the hub grows by a certain amount; the amount used was the least distance between the actual radial point and the other web threads (frame and support threads). This continues until the radial thread is close enough to another thread. When this happens, this closest point is considered the intersection point between the radial line and another thread. This method uses only about 3 or 4 iterations at most to find the intersection point for each radial thread.

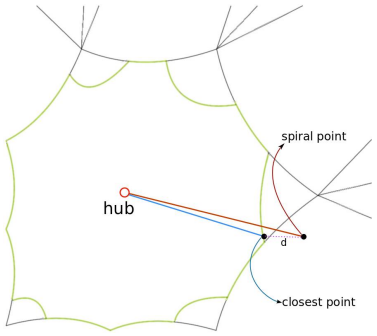


Fig. 9: A representation of the algorithm to check if a spiral points exceeds the boundaries of the orb-web. To verify that, a *closest point* from the *spiral point* is found, considering the points of the green hull. After this, the distance from the hub from both points is compared. If the spiral point is closer from the hub, it is inside the hull and must be draw, otherwise not. In this example, the spiral point should not be drawn.

5) Spiral Thread: Here we place the sticky spiral of the spider orb-web. The spiral does not start in the center, as seen in real orb-webs. The position may vary for different species of spiders, and with the size of the web. Hence, we used a distance of about 10% of the distance from the hub to the furthest frame thread plus some variation of 3%.

To make the process of constructing the spiral as simple as possible, we considered the spiral as being an Archimedean Spiral (also known as an arithmetic spiral), described by the polar equation $r = a + b\theta$. This spiral has the property that successive turnings keep a constant separation distance from each other (equals to $2\pi b$ if θ is in radians). This property makes this spiral suitable for representing the spiral of a spider orb-web, as already studied in [5]. Moreover, to make the spiral web more realistic, we inserted some disturbance when computing the spiral. This can be done by slightly changing the values of a and b in the polar equation. The variation of the disturbance is presented in Table I.

To draw the spiral part, we have to check that it does not exceed the boundaries of the orb-web. To check this, first we get the list of frames or support threads, depicted by the green segments in Figure 9; next, we find in that list the closest point from the actual spiral segment. Now, we only need to compare the distance from the hub between the actual spiral segment and the closest point found. If the distance of the spiral segment is larger, it is out of the limits of the web and must not be drawn. The algorithm continues to compute the upcoming spiral segments and once the spiral enters again the boundaries of the orb-web, it starts to drawn it again.

6) Web Wire Rendering: To render wires, a primitive that would be more suitable is the cylinder, due to its geometric similarity with real threads. In [9], the authors even extended this primitive, creating what they called a *Spherinder*, which is a cylinder with two hemispheres at either end in place of the traditional flat caps of a regular cylinder. These way two

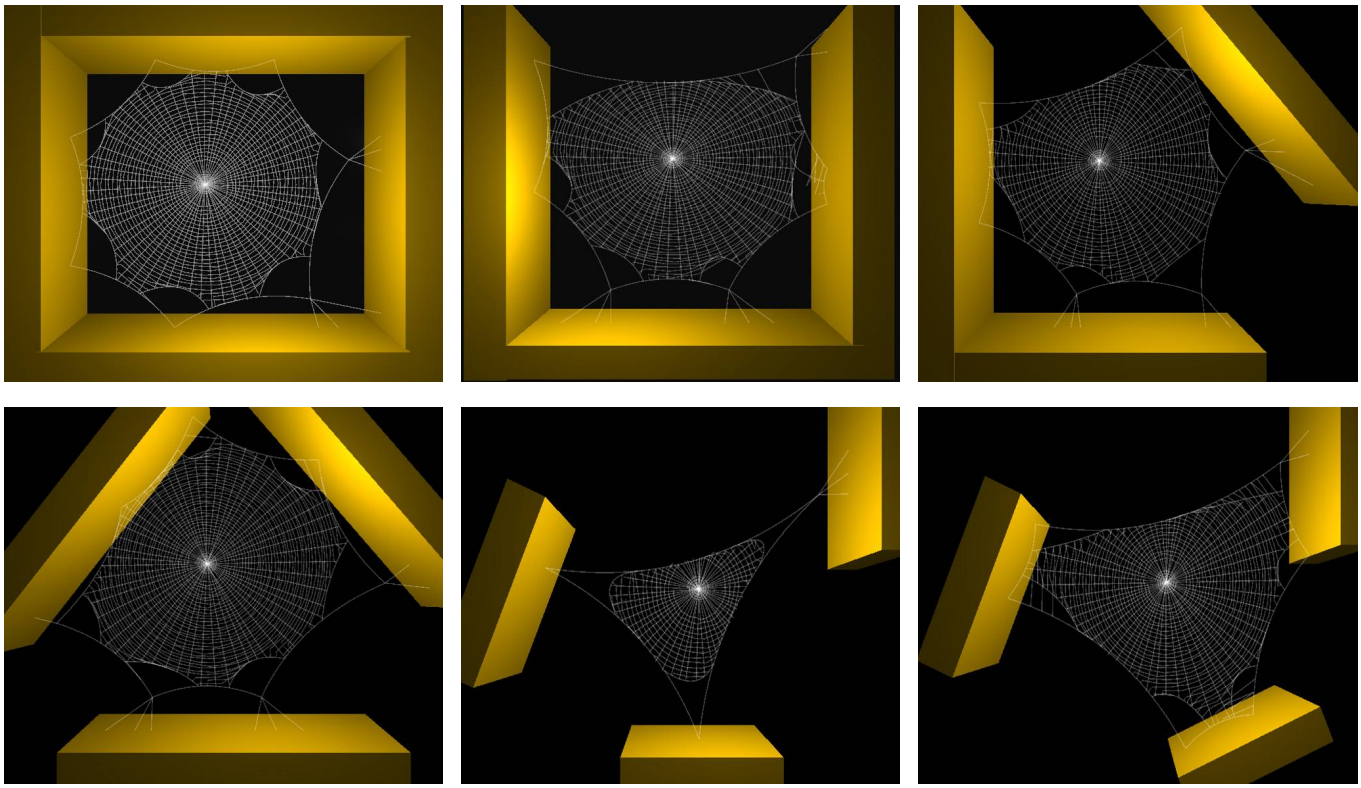


Fig. 10: In each of these examples, the surrounding objects were slightly modified to show how the orb-web can automatically adapt to different configurations of environment.

cylinders would overlap more smoothly.

For real-time goals, we considered these primitives costly. A cylinder, for instance, roughly approximated by eight faces would need 16 vertices to represent. Hence, the shape we decided to use is simpler: a plane consisting of 4 vertices each segment. In addition, after the first plane, any other plane connecting to it will only need two more vertices. The plane segment is defined by its position, thickness, and a normal. The normal orientation used is the same web plane normal, defined in Figure 5. Therefore, given a list of points, we create the plane segments based on simple geometric calculations (Figure 11). The sampling along the web may vary depending on the desired quality of the Bézier curves. The more points used to represent the curve, smoother the wires should appear.

Finally, the wire is texturized to achieve a more realistic appearance. The texture used is just a white 32×32 image, with transparent borders to avoid aliasing.

III. Results

In this section, we present the results of our final spider orb-web images rendered in real-time. The computer used to run the tests was a Core i7 2600 3.4GHz with 12GB of DDR3 RAM and a GeForce GTX580. Our algorithm was implemented using Unity3D [12], a powerful game engine with many graphical cutting-edge technology aimed for real-time graphics. The synthetic scenes where the orb-webs were inserted used only basic lighting and texture mapping.

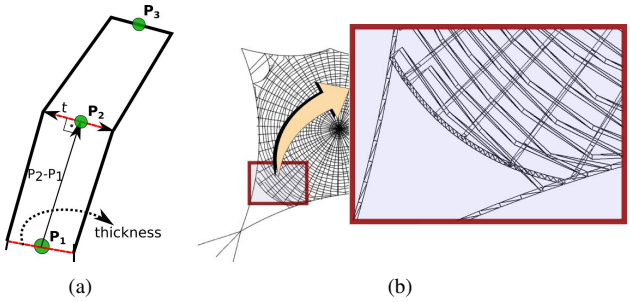


Fig. 11: (a) Given 3 points P_1, P_2 and P_3. Plane segments connecting them can be calculated by finding a vector $\vec{t}$, perpendicular to $(P_2 - P_1)$. This can be achieved by computing the cross product between the normal to the plane and $(P_2 - P_1)$. Then the resultant vector $\vec{t}$ is normalized and multiplied by the plane thickness, to give its correct size. (b) A close-up of a web rendered in wireframe allows us to see the many connected planes that form the wires.

The most essential feature of our algorithm is the web adaptation. It is very important that the web can assume a different configuration depending on the environment it was inserted. Figure 10 depicts how our algorithm handle changes in the surrounding objects. First, for the web to exist, it must found at least 3 anchor points. If these points can define a

TABLE II: Average running time (in miliseconds) of each phase of the algorithm to build the webs from Figure 12.

Algorithm Phases	Average Running Time
Find anchor points	0.513 ms
Build frame thread	1.132 ms
Build support thread	0.303 ms
Build radial threads	41.780 ms
Build spiral threads	314.601 ms
Total Time	**358.329 ms**

plane in the space (not collinear), the web can be build. It can be seen that for every different configuration of the scene objects, the web finds a way to fit in the environment. This assures that the algorithm can generate many different patterns of spider orb-webs.

The next results are a comparison of our final web rendering with some real spider orb-webs. For a more fairly comparison, we blended our orb-web rendered images into real pictures of natural landscapes. The pictures of real spider orb-webs are shown in Figure 2 and our results are shown in Figure 12. The synthetic orb-webs seem to approximate the shapes of real spider-webs and are visually appealing. Although, it can be seem that real spider orb-webs still have greater variability – especially considering the frame and support threads –, this occur due to the simplifications of our model considering biological variables. Another reason for that is that spiders are often repairing their web to increase stability, which can lead to patterns that are difficult to mimic using our algorithm. Figure 2 (bottom row), shows some complex variations of frame threads and support threads that occur in real orb-webs.

Web wires are rendered using a simple illumination model. Since webs are represented as 3D meshes, the rendering can be improved to simulate more realistic types of thread. Another advantage of using a 3D mesh is that more advanced rendering techniques can be used, such as shader effects.

Orb-webs are 3D models defined by vertices and faces. Figure 12 shows the number of vertices and faces for each one of the synthetic spider web. It can be seem that the vertex values do not vary much, averaging 1882 approximately. Such low number of polygons can be easily handled in real-time applications. In addition, the technique is efficient and can build orb-webs in real-time, as shown by the average running time for the algorithm in Table II.

IV. Conclusion and Future Work

In this paper, we presented an approach to construct self-adaptive spider orb-webs in a procedural manner, eliminating the time-consuming task of modeling organic objects. This approach has proved efficient and outputs good quality orb-webs that run in real-time. Since our concern is mainly with the final appearance and the rendering quality, we do not take into consideration the directly influence of biological variables, such as, wind, temperature, humidity, silk supply, among others, seeing that would end up with a much more computationally expensive model. On the other hand, these simplifications limit the variability that may occur in the structure of the orb-webs, even inserting an amount of randomness to the parameters.

For future work, a more complex model can be achieved by elaborating new patterns of threads, mainly the frame and support threads. To prevent a decrease in the performance in real-time applications, many level of details could be implemented, leading to more realistic models up close to the viewer. Furthermore, the algorithm could be generalized to simulate more aged orb-webs, considering broken wires, for instance. This surely would give a more realistic appearance for the web. Another great improvement would be to apply shader effects to better simulate the properties of the wire material, like the spectral dispersion of light. In addition, effects such as blooming and anti-aliasing would give a better final rendering image. Another interesting extension would be to work with physically based animation models. We believe it would be interesting to simulate animations caused, for instance, by the wind, that affects all the web structure. Moreover, wire tension simulations could be valuable for many different research areas [13], such as biology or architecture, since the spider silk has a unique combination of strength and elasticity that make it one of the toughest known materials.

Acknowledgment

The second author acknowledges the partial financial support from CNPq through grant 478730/2012-8.

References

[1] R. Fedkiw, "Simulating natural phenomena," in *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer New York, 2003, pp. 461–479.

[2] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '98. ACM, 1998, pp. 275–286.

[3] B. Lintermann and O. Deussen, "Interactive modeling of plants," *Computer Graphics and Applications, IEEE*, vol. 19, no. 1, pp. 56–65, 1999.

[4] F. Vollrath, M. Downes, and S. Krackow, "Design variability in web geometry of an orb-weaving spider." *Physiol Behav*, vol. 62, no. 4, pp. 735–43, 1997.

[5] W. Eberhard, "Computer simulation of orb-web construction," *American Zoologist*, vol. 9, no. 1, pp. 229–238, 1969.

[6] N. Gotts and F. Vollrath, "Artificial intelligence modelling of web-building in the garden cross spider," *Journal of Theoretical Biology*, vol. 152, no. 4, pp. 485–511, 1991.

[7] T. Krink and F. Vollrath, "Analysing spider web-building behaviour with rule-based simulations and genetic algorithms," *Journal of Theoretical Biology*, vol. 185, no. 3, pp. 321–331, 1997.

[8] E. P. Lafortune, "Realistic modeling of spider webs," http://www.lafortune.eu/publications/spiderwebs.html, 1997, [Online; accessed 01-July-2013].

[9] J. G. Bridgette Wiley and M. Fecho, "Spectral spiderweb," https://sites.google.com/site/mlfecho/cse168-rendering-project, 2011, [Online; accessed 01-July-2013].

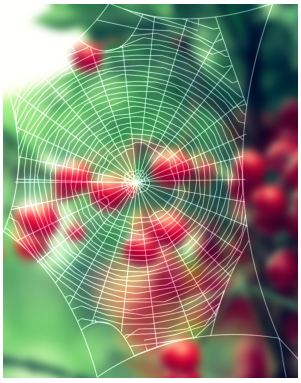
[10] F. Vollrath, "Spider webs and silks," *Scientific American*, vol. 266, no. 3, pp. 70–76, 1992.

[11] F. Vollrath and W. Mohren, "Spiral geometry in the garden spider's orb web," *Naturwissenschaften*, vol. 72, no. 12, pp. 666–667, 1985.

[12] Unity3d, *version 3.5*. San Francisco: Unity Technologies, 2012.

[13] E. Wirth and F. Barth, "Forces in the spider orb web," *Journal of Comparative Physiology A*, vol. 171, no. 3, pp. 359–371, 1992.

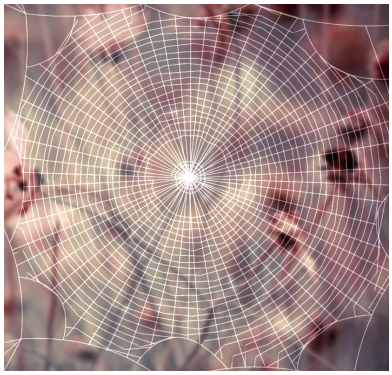
(a) 1852 vertices.



(b) 1852 vertices.

(c) 1678 vertices.

(d) 1930 vertices.



(e) 2064 vertices.

(f) 1886 vertices.

(g) Three webs from left to right, with 1934, 1712 and 1844 vertices, respectively.

Fig. 12: Spider webs generated procedurally using our algorithm in real-time. The rendered image of the orb-web was blended into real photographs to given a more realistic effect.