Tuning Manifold Harmonics Filters

Thomas Lewiner*, Thales Vieira†, Alex Bordignon*, Allyson Cabral*, Clarissa Marques*, João Paixão*, Lis Custódio*, Marcos Lage*, Maria Andrade*, Renata Nascimento*, Scarlett de Botton*, Sinésio Pesco*, Hélio Lopes*, Vinícius Mello‡, Adelailson Peixoto† and Dimas Martinez†.

* Matmidia Lab, Department of Mathematics, PUC-Rio, Rio de Janeiro, Brazil.
† Institute of Mathematics, UFAL, Maceió, Brazil.
‡ Institute of Mathematics, UFBA, Salvador, Brazil.

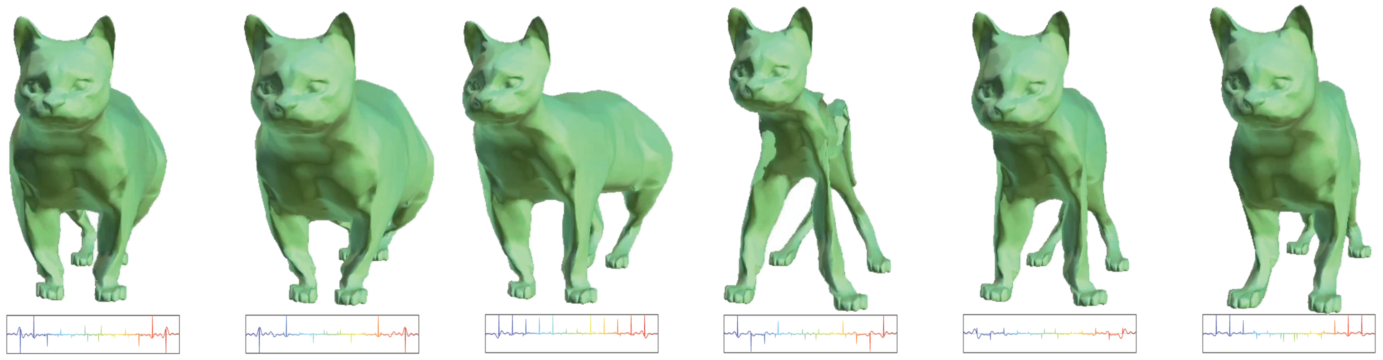


Fig. 1. Music visualization by deforming a 3d model according to the music amplitudes.

***Abstract*—There are several techniques for automatic music visualization, which are included with virtually any media player. The basic ingredient of those techniques is spectral analysis of the sound, used to automatically generate parameters for procedural image generation. However, only a few music visualizations rely on 3d models. This paper proposes to use spectral mesh processing techniques, namely manifold harmonics, to produce 3d music visualization. The images are generated from 3d models by deforming an initial shape, mapping the sound frequencies to the mesh harmonics. A concise representation of such frequency mapping is proposed to permit for an animated gallery interface with genetic reproduction. Such galleries allow the user to quickly navigate between visual effects. Rendering such animated galleries in real-time is a challenging task, since it requires computing and rendering the deformed shapes at a very high rate. This paper introduces a direct GPU implementation of manifold harmonics filters, which allows to display animated gallery.**

***Keywords*-Manifold Harmonics; Sound Visualization; Geometry Processing; GPU; Design Galleries;**

I. Introduction

The illustration of music became a necessary part of the audio industry. While video clip is now a complete part of a song production, almost any computer program that renders sound content offers several visualizations. Most audio visualization techniques rely on Fast Fourier transforms that extract the harmonic amplitudes of the sound samples. These amplitudes serve as parameters to algorithms that generate beautiful or exciting images in real time, using procedural techniques from simple digital peak meters to psychedelic dynamical systems. We propose to generate images obtained by deforming an initial discrete 3d model (see Fig. 1).

Since the sound analysis relies on sound harmonics, a natural approach is to use geometric harmonics to deform the 3d model. A definition of such geometric harmonics, called *manifold harmonics* has been recently proposed by Vallet and Lévy [1]. Amplifying some harmonics of a given mesh leads to coherent deformations, in the sense that filtering low frequencies actually deforms the global shape of the mesh, while altering high frequencies changes its details.

However, using manifold harmonics for sound visualization is a two-fold challenge: First, manipulating the amplitudes of each harmonic is a delicate task, since nearby frequencies have very different and dramatic impacts on the shape. Second, the deformation must be rendered in real-time to keep synchronized with the music. In this paper, we propose to model the mapping of sound harmonics amplitudes to manifold harmonics amplitude using a design gallery with genetic reproduction, in a way similar to what is commonly done in volume visualization [2].

The use of gallery turns the second challenge even more difficult, since an animated gallery requires to compute and render several deformations of the initial mesh for each frame. We propose here a direct GPU implementation of the manifold harmonics filter that copes with such requirements. For models containing around 50,000 vertices, we can render a gallery of 12 animated deformations in real-time.

II. Related work

There are several techniques for automatic music visualization, as one can see on virtually any media player. However, only a few of them use 3d models. To the best of our knowledge, the closest work relating sound and 3d objects come from granular mechanics simulation [3], starting back to the studies of vibration modes [4], *Modal Analysis* [5] became a very important tool in the understanding of mechanical structure responses. Modal Analysis was first introduced to Computer Graphics by Pentland and Williams [6], where they used it to simulate deformations in non-rigid objects from a sound signal. A reduced version of such simulations has recently been brought to real-time through a GPU implementation, but only using the first few vibration modes [7]. In this paper, we propose a music visualization scheme instead of a mechanical simulation, and achieve real-time performance in a complete spectral processing system. Note that a process inverse to this objective, i.e. creating audio content from a 3d animation, has ben proposed by O'Brien *et al.* [8]

Since the seminal work of Taubin [9], several approaches have been proposed to adapt signal processing techniques to discrete surfaces. Among those, spectral processing has gained a lot of attention [10]. Those methods rely on defining an equivalent for Fourier harmonics (basically sine and cosine) as eigenvectors of Laplace-like operators. Among those works, Vallet and Lévy proposed a manifold harmonics adapted to mesh edition [1]. This work motivated several applications in connected fields: spectral mesh deformation [11], mesh watermarking [12], [13] and shape analysis [14], [15]. In this paper, we use manifold harmonics filters, and propose a fast GPU implementation of spectral filtering to obtain real-time deformations.

III. Manifold Harmonics

In this section, we will recall the basics of manifold harmonics following the original work of Vallet and Lévy [1].

The idea behind manifold harmonics is to transpose usual Fourier edition to 3d meshes. In Fourier analysis, a functional basis $\mathbf{h}_\omega(t) = \mathbf{e}^{-2\pi \cdot i \cdot \omega \cdot t}$ of so-called *harmonics* is used to decompose an input signal $f(t)$ into a combination of those harmonics:

$$f(t) = \int_{\mathbb{R}} \tilde{f}(-\omega) \cdot \mathbf{h}_\omega(t) \mathbf{d}\omega \quad with \quad \tilde{f}(\omega) = \int_{\mathbb{R}} f(t) \cdot \mathbf{h}_\omega(t) \mathbf{d}t \, .$$

A. Laplace harmonics

The main observation is that those harmonics $\mathbf{h}_\omega$ are the eigenvectors of the differential Laplace operator Δ_∂:

$$\Delta_\partial (\mathbf{h}_\omega) \equiv \frac{\partial^2 \mathbf{h}_\omega}{\partial t^2} = \lambda_\omega \cdot \mathbf{h}_\omega \quad with \quad \lambda_\omega = -4\pi^2 \, \omega^2 \, .$$

To transpose such decomposition on a mesh, a natural option is to look for the eigenvectors of a discrete Laplace operator. Vallet and Lévy derive a Laplace-De Rham operator from Discrete Exterior Calculus [1]. On the vertices of a mesh, this operator turns out to be linear, and can thus be expressed as an $n \times n$ matrix Δ, where n is the number of vertices of the mesh. Its coefficients Δ_{ij} are zero if vertices i and j are not adjacent, and otherwise:

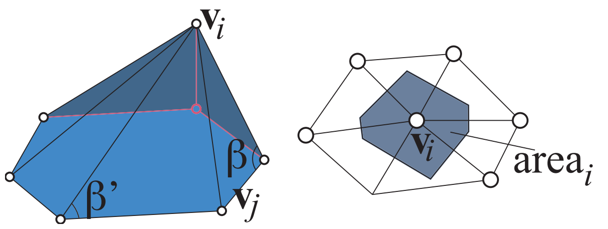


Fig. 2. Geometric elements for the coefficients of the discrete Laplace operator.

$$\Delta_{ij} = -\frac{cot\left(\beta_{ij}\right) + cot\left(\beta'_{ij}\right)}{\sqrt{area_i \cdot area_j}} \, , \quad \Delta_{ii} = -\sum_j \Delta_{ij} \, ,$$

where $area_i$ is the area of the restricted Voronoi region of vertex i, and the angles β_{ij} and β'_{ij} are opposite to the edge between i and j (see Fig. 2).

B. Manifold harmonics transform

With a slight rescaling of the areas $area_i$ [1], the matrix of this discrete Laplace operator Δ is symmetric, and can thus be diagonalized, obtaining an orthonormal basis eigenvectors of $\mathbf{H}_k \in \mathbb{R}^n$ associated to eigenvalues $\Lambda_k \in \mathbb{R}$. Since this is a basis in $\mathbb{R}^n$, any function $F : i \in \{0, \ldots, n-1\} \mapsto \mathbb{R}$ defined on the vertices of the mesh can be decomposed on this basis:

$$F(i) = \sum_{k=0}^{n-1} \tilde{F}(k) \cdot \mathbf{H}_k \quad with \quad \tilde{F}(k) = \sum_{i=0}^{n-1} F_i \cdot \mathbf{H}_k \, .$$

Using the analogy with Fourier analysis, the frequency associated with Λ_k is $\sqrt{\Lambda_k}$, and we consider that the frequencies are ordered: $\Lambda_0 \leq \Lambda_1 \leq \ldots \leq \Lambda_{n-1}$.

C. Filtering

The signal $F(i)$ above is thus expressed as a combination of harmonics $\mathbf{H}_k$, with respective amplitudes $\tilde{F}(k)$. A linear filter can then be expressed by amplifying each harmonic $\mathbf{H}_k$ by a factor $\varphi(k)$. The filtered signal $F_\varphi(i)$ is then given by:

$$F_\varphi(i) = \sum_k \varphi(k) \cdot \tilde{F}(k) \cdot \mathbf{H}_k \, .$$

Since we are here interested in deforming the mesh, we will consider the signal $F(i)$ to be the coordinates $x(i), y(i), z(i)$ of vertex i. We therefore get three harmonic amplitudes $\tilde{x}(k), \tilde{y}(k), \tilde{z}(k)$ for each frequency k. Since the mesh is not *a priori* aligned, we will filter all the three coordinates with the same amplification φ. Finally, since high frequencies correspond to very small perturbations, appearing as noise, we will only filter using the lowest $\#k$ frequencies:

$$F_\varphi(i) = \sum_{k=0}^{\#k-1} \varphi(k) \cdot \tilde{F}(k) \cdot \mathbf{H}_k + d_i \quad with \quad d_i = \sum_{k=\#k}^{n-1} \tilde{F}(k) \cdot \mathbf{H}_k \, .$$

The coefficients d_i can be computed at preprocessing.

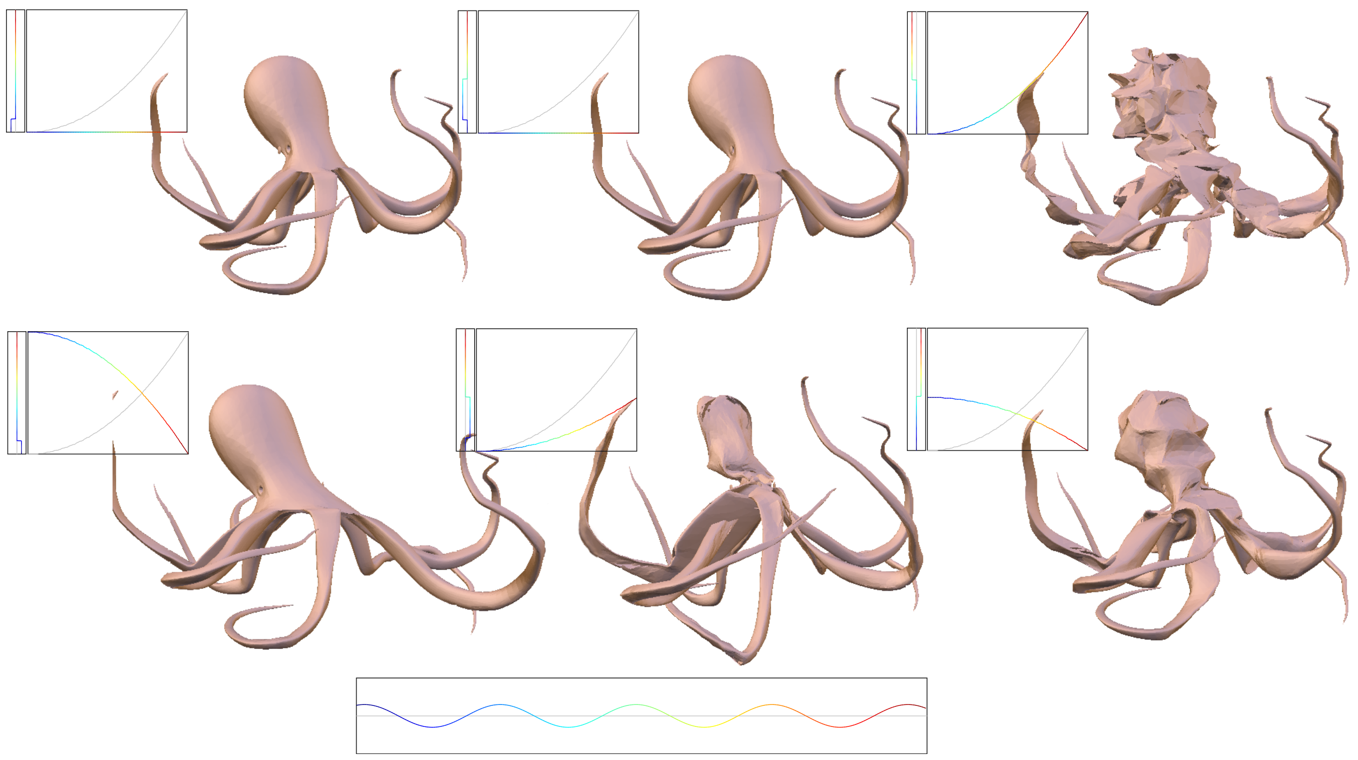


Fig. 3. Initial gallery on an octopus model, with the corresponding transfer and amplification functions. The frequency input $\tilde{f}$ is drawn at the bottom.

IV. Tuning Manifold Harmonics

We want to apply manifold harmonics filters to illustrate signals $f(t)$ such as audio content. Since manifold harmonics filters are very sensitive, eventually leading to large deformation for small variations of the filter, the transfer of signal harmonic amplitudes $\tilde{f}(k)$ to the manifold harmonic amplitudes $\varphi(k)$ would require a very careful edition if done manually. In this section, we introduce a simple design model for such mapping. This design allows a gallery interface [16] with genetic reproduction [2], which permits the user to quickly navigate between mappings (see Fig. 3).

A. *Mapping to manifold harmonics filters*

We want to design a filter $\varphi(k)$ from the harmonic amplitudes $\tilde{f}(\omega)$ of an input signal, where the dependency $\varphi(k) = \Phi(\tilde{f})(k)$ of φ from $\tilde{f}$ is not necessarily linear. Moreover, the number of frequencies $\#\omega$ computed from the signal may differ from the number of harmonics $\#k$ of the mesh. We decompose this mapping in two steps: a frequency transfer function $t : \omega \mapsto k \in \{0, \ldots, \#k - 1\}$ combined with an amplification function $a : k \mapsto a(k) \in \mathbb{R}$ applied on the manifold harmonic amplitudes.

We want each harmonic of the mesh to receive contributions from different harmonics of the signal, so that a musical instrument, which covers different frequencies, could be mapped to a single manifold harmonic. Therefore, the transfer function maps sound frequencies ω to mesh frequencies k, and a mesh frequency k will receive contributions from all the sound frequencies in $t^{-1}(\{k\})$. We propose a harmonic mapping $\Phi_{t,a} : \tilde{f} \mapsto \varphi$ as (see Fig. 4):

$$\Phi_{t,a}(\tilde{f})(k) = a(k) \cdot \left(\sum_{\omega \in t^{-1}(\{k\})} \tilde{f}(\omega) \right) + 1 \ .$$

By adding one, we maintain the usual intuition of amplification: amplifying all the mesh frequencies to 0 (*i.e.* $a \equiv 0$) does not deform the mesh. Note that, since the harmonic amplitudes of the sound may be negative, the amplification a may also be negative.

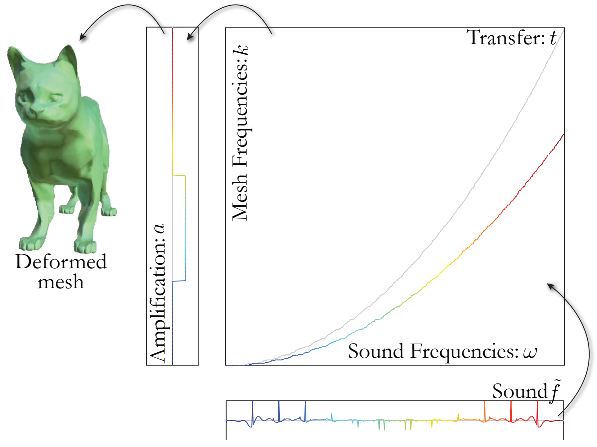


Fig. 4. Filter design as the composition of a frequency transfer function t and an amplification function a (drawn vertically). The grey curve for the transfer corresponds to the direct mapping from $\omega \cong \sqrt{\Lambda_{t(\omega)}}$.

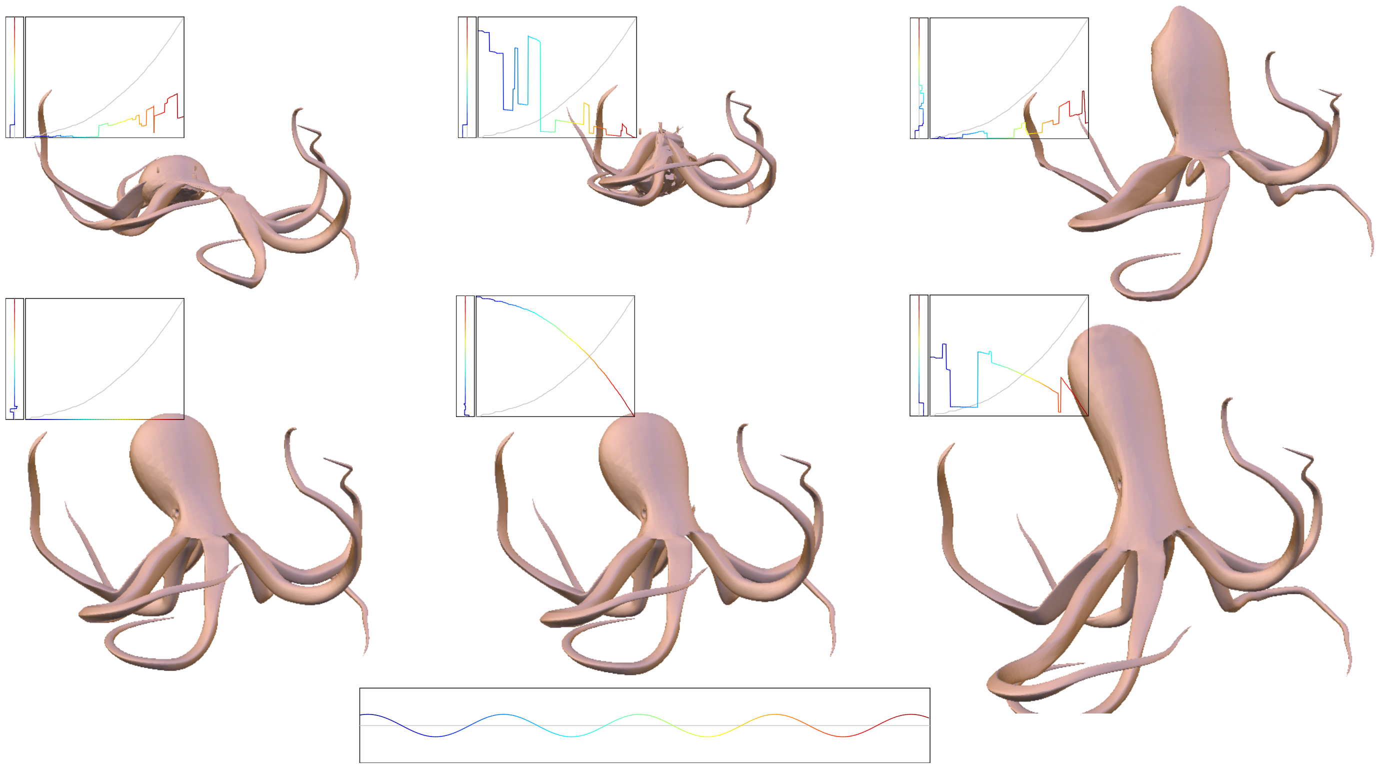


Fig. 5. Gallery after one reproduction from the 1^{st}, 4^{th} and 5^{th} items of Fig. 3 in reading order, with the reproduced transfer and amplification functions.

B. Tuning through design galleries

The filter design above gives a concise representation of the harmonic mapping $\Phi_{t,a}$ from sound harmonic amplitudes to manifold harmonic amplitudes. Indeed $\Phi_{t,a}$ is represented as two vectors: $t \in \mathbb{N}^{\#\omega}$ is an integer vector of size $\#\omega$, and $a \in \mathbb{R}^{\#k}$ is a real vector of size $\#k$. This allows to easily mix two harmonic mappings by combinations of those vectors. Using vocabulary from genetic algorithms, the harmonic mapping $\Phi_{t,a}$ is represented by two chromosomes a and t, which can reproduce by combination.

This leads to a direct design gallery interface, where different harmonic mappings are proposed to the user, who can select the ones he likes. From this selection, a new gallery is generated using genetic reproduction, until the user chooses only one harmonic mapping, as explained in the next section. The following section will detail the initial gallery creation. The harmonic mapping can then be directly edited from the two curves of t and a.

C. Reproduction

The reproduction generates a new gallery of S harmonic mappings from a selection of old mappings. To do so, S pairs of distinct selected old mappings are randomly chosen. Each pair is then combined into a new mapping as follows.

Since the frequency transfer and amplification functions t and a have complementary effects, we reproduce them independently. This also reduces the initial gallery size, as explained in the next subsection. In practice, this means that we first decide if we combine the frequency transfer functions of the pair using a $\frac{1}{2}$-Bernoulli trial ("heads or tails"). We decide in a similar manner if the amplification functions will be combined.

The combination of the frequency transfer functions t' and t'' of the pair is done as follow. First we randomly choose an integer value n_k^0, as a geometric random variable in $\{1, \dots \#k\}$, and a random real value w^0 uniformly in $[0, 1]$. We then set the first n_k^0 coefficients of vector t as the first n_k^0 coefficients of $w^0 \cdot t' + (1 - w^0) \cdot t''$. We choose again random values $n_k^1 \in \{1, \dots \#k\}$ and $w_1 \in [0, 1]$, and clamp n_k^1 to ensure $n_k^0 + n_k^1 \leq \#k$ (the geometric random process intends to reduce the effect of this clamping). We then set the following n_k^1 values of t as above, and repeat until completing all the frequencies. We perform the same operations for the amplifications (see Figs. 3 and 5).

This combination method avoids producing combination that varies too quickly, as compared to randomly choosing real values w at each frequency.

D. Gallery initialization

We generate an initial gallery (see Fig. 3) that could theoretically generate any harmonic mapping by the above reproduction. Since the reproduction of the frequency transfer and amplification are independent, we can use the S elements of the initial gallery to span the frequency transfer functions and the same S elements to span the amplification functions. This reduces the size of the initial gallery, although it generally requires one more reproduction to get interesting mappings.

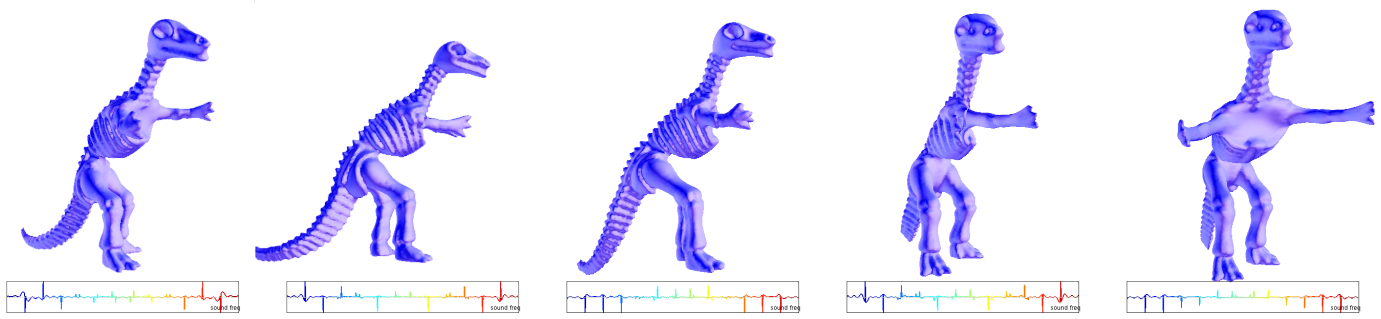


Fig. 6. Frames of pop music visualization using the dinosaur model.

The first frequency transfer function is the direct mapping:

$$t_{ini}(\omega) = \min\{k \text{ such that } \frac{\omega}{\omega_{\#\omega}} \leq \sqrt{\frac{\Lambda_k}{\Lambda_{\#k}}}\} \ .$$

This expression ensures that, if there exist a unique k such that $\frac{\omega}{\omega_{\#\omega}} = \sqrt{\frac{\Lambda_k}{\Lambda_{\#k}}}$, then $t_{ini}(\omega) = k$. This function maps the sound low (resp. high) frequencies to the mesh low (resp. high) frequencies. The function $t_{rev} = \#k - t_{ini}$ maps high sound frequencies to low ones, and *vice versa*.

Usually, altering the low frequencies of the mesh give more visible effects. We therefore define the frequency transfer functions of the initial galleries as condensed transfers towards the low frequencies: $t(\omega) = t_{ir}(\alpha \cdot \omega)$, where t_{ir} is either t_{ini} or t_{rev} and $\alpha \in \{0, 1, \frac{2}{S}, 2\frac{2}{S}, 3\frac{2}{S}, \ldots\}$. The first value $\alpha = 0$ is a constant mapping to the lowest and highest frequency. It is included to guarantee that any transfer function can be generated by combinations.

The amplification functions of the initial gallery are simple band-pass filters, with positive or negative factors. The interval of manifold harmonic frequencies $\{0, \ldots, \#k - 1\}$ is divided in intervals I_α, for $\alpha \in \{\frac{S}{2}, 2\frac{S}{2}, 3\frac{S}{2}, \ldots\}$. After we define the amplification function for the first half of the gallery $a_\alpha(k) = M$ if $k \in I_\alpha$, and $a_\alpha(k) = 0$ otherwise, where M is the maximal amplification factor. The other half is defined similarly using $-M$. If the sound amplitudes are normalized to $[-1, 1]$ and the if the mesh is reasonably smooth, the order of magnitude of M is 5,000. Since we try to emphasize the low frequencies, we define the intervals $I_\alpha = [\alpha^2, (\alpha + \frac{2}{S})^2]$.

V. Making It Real-Time

The main challenge for the above interface to work with sound signals is to compute and render the deformation of each gallery element synchronously with the music (see Fig. 6). If we have S elements in the gallery, each of which is a mesh of n vertices with $\#k$ manifold frequencies and $\#\omega$ sound frequencies, a single frame represents $O(S \cdot \#\omega \cdot \#k \cdot 3n)$ operations! (The $\#\omega$ factor comes from the evaluation of $\Phi_{t,a}$). We therefore propose a GPU implementation of the manifold harmonics filter, while the manifold harmonics decomposition is pre-computed in CPU.

```
uniform sampler1D x̃ỹz̃;
uniform sampler2D d_xyz;
uniform sampler3D H_k;
uniform sampler1D φ;
uniform float δk;

void main() {
    vec3 texcoord = gl_TexCoord[0].stp ;
    vec3 pos = texture2D(d_xyz,texcoord.st).xyz ;
    for( float k=0.0; k ≤ 1.0; ) {
      texcoord.p = k ;
      vec4 H = texture3D(H_k, texcoord);
      vec4 f = texture1D(φ, k);
      vec3 x̃ỹz̃_0 = texture1D(x̃ỹz̃, k).xyz ; k += δk ;
      vec3 x̃ỹz̃_1 = texture1D(x̃ỹz̃, k).xyz ; k += δk ;
      vec3 x̃ỹz̃_2 = texture1D(x̃ỹz̃, k).xyz ; k += δk ;
      vec3 x̃ỹz̃_3 = texture1D(x̃ỹz̃, k).xyz ; k += δk ;
      pos += f[0] * H[0] * x̃ỹz̃_0 + f[1] * H[1] * x̃ỹz̃_1 +
             f[2] * H[2] * x̃ỹz̃_2 + f[3] * H[3] * x̃ỹz̃_3 ;
    }
    gl_FragColor.rgb = pos.xyz ;
}
```

Fig. 7. GLSL fagment shader for the manifold harmonics filter.

A. GPU implementation

For the sake of portability, we chose to use GLSL [17] as GPU language. The manifold harmonics filters actually require a single fragment shader, which computes the filtering F_φ for each coordinate x, y, z (see Section III-C), together with a render-to-vertex-buffer mechanism [18].

Data textures: The manifold harmonics is sent to the GPU as textures: a texture x̃ỹz̃ containing the harmonic amplitudes $\tilde{x}(k), \tilde{y}(k), \tilde{z}(k)$ of the original mesh, a texture $\mathsf{d}_{\mathsf{xyz}}$ containing the sum of high frequencies contributions for each coordinate (see Section III-C) and a texture H_k containing the manifold harmonics eigenvectors. The filter φ must be sent to the GPU at each gallery element of each frame. Since the computed φ has a smaller size than the sound frequencies $\tilde{f}$ and the t and a vectors, we compute φ on the CPU and send it as a 1D texture ϕ.

Texture storage: All the textures are stored using 32 bits floats to keep the precision of the vertices coordinates. Since the number of vertices of the mesh is usually higher than the maximal texture size for 1D textures, we use two texture coordinates in $\{0 \ldots \lceil \sqrt{n} - 1 \rceil\}$ as vertex indexes. The high frequency contributions $\mathsf{d}_{\mathsf{xyz}}$ are stored as a 2D RGB texture of size $\lceil \sqrt{n}\ \rceil \times \lceil \sqrt{n}\ \rceil$, where the coordinates are mapped to the RGB color components.

Since the number of manifold frequencies $\#k$ kept fits in a texture row, the original harmonic amplitudes texture $\tilde{\mathsf{x}}\tilde{\mathsf{y}}\tilde{\mathsf{z}}$ are stored as a 1D RGB texture of size $\#k$, where the $\tilde{x}, \tilde{y}, \tilde{z}$ components are mapped to RGB.

Finally, the scalar data H_{k} and ϕ of the manifold harmonics eigenvector n coordinates and the filter can be stored in the RGBA components to optimize space: ϕ is then a $\lceil \frac{\#k}{4} \rceil$ 1D and H_{k} an $\lceil \sqrt{n}\ \rceil \times \lceil \sqrt{n}\ \rceil \times \lceil \frac{\#k}{4} \rceil$ 3D RGBA textures.

Fragment shader for the filter: When all the above textures are bound, the rendering of a single square of size $\lceil \sqrt{n}\ \rceil \times \lceil \sqrt{n}\ \rceil$ will call the fragment shader for each of the vertex index and compute the new vertex positions as frame color (see Fig. 7). The fragment shader renders to a frame buffer containing the filtered vertex coordinates, which is then copied to the vertex buffer inside the GPU [18]. The shader receives a uniform variable which is the normalized increment $\delta k = \frac{1}{4(\#k-1)}$ for manifold frequency iteration inside normalized texture coordinates.

B. Complementary effects

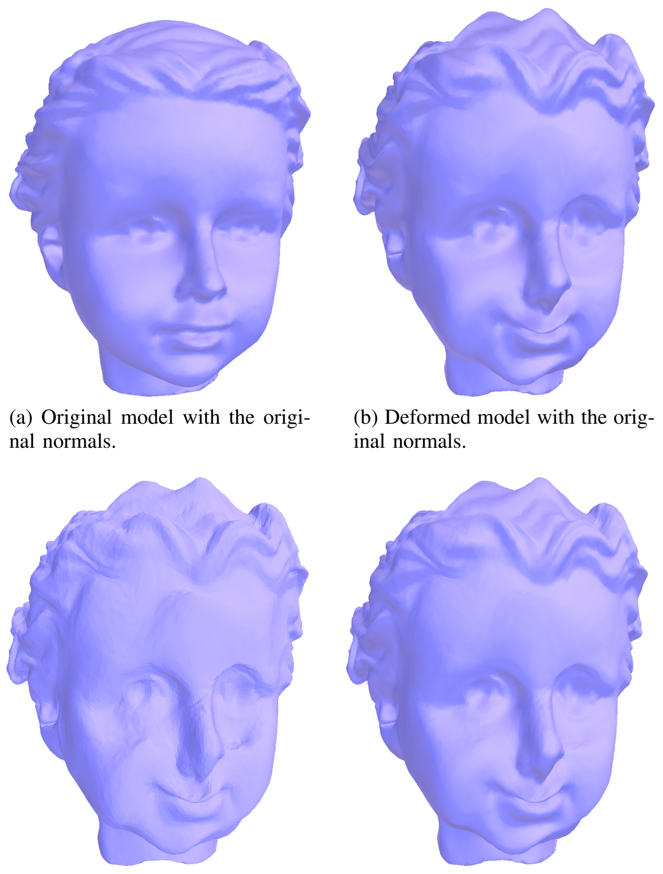
Normal enhancement: The previous method updates the vertex positions, but not the normal. Since a second render-to-vertex-buffer would be too costly, we use a geometry shader that computes, for each triangle, a constant normal. This normal is used in a per-pixel lighting via fragment shader. However, the constant normal per triangle leads to flat shading. To obtain smoother result, we average, in the geometry shader, the constant normal of the triangle with the original normal of the vertex (see Fig. 8).

Harmonic mapping re-use: The gallery interface allows to quickly navigate between all the possible harmonic mappings within our proposed design. If a harmonic mapping gives a very exciting effect, it would be nice to be able to re-use it on other models. The main obstacle is that the number of manifold frequencies $\#k$ may differ from model to model. We can work around this problem by normalizing the image values of t to a constant interval $[0,1]$: $\bar{t}(\omega) = \frac{t(\omega)}{\#k}$, and adapt the definition of $\Phi_{t,a}$ to $\Phi_{\bar{t},a}(\tilde{f})(k) = a(k) \cdot \left(\sum_{\omega \in \bar{t}^{-1}(\{\frac{k}{\#k}\})} \tilde{f}(\omega) \right) + 1$.

Beat detection: Until now, the whole deformation of the mesh is seen from a single point of view with a constant lighting. We propose to use those degrees of freedom to transpose global sound feature, such as beat. We implemented a simple beat detection [19], and at each detected beat we randomly choose to rotate the model or the light positions.

C. Implementation details

We used the Scalable Library for Eigenvalue Problem Computations (`SLEPc`) [20] software package to compute the first $\#k$ manifold harmonics eigenvalues and eigenvectors. We use the Compact Half Edge [21] data structure to represent the model mesh. The proposed shaders require an `OpenGL` 2.x compatible card [17]. Finally, we use `FFmpeg` [22] for sound decompression and `OpenAL` [23] for stereo sound rendering in a separate thread. A nice tutorial for such sound configuration can be found at kcat.strangesoft.net/openal.html.



(a) Original model with the original normals.

(b) Deformed model with the original normals.

(c) Deformed model with the deformed, flat normals.

(d) Average normals with 40% deformed normals.

Fig. 8. Normal enhancement for the deformed model.

VI. Results

We experimented the proposed filter design with gallery interface to check the feasibility of such approach. The actual validation of the interface is beyond the scope of this paper. However, our proposal is able to provide an animated gallery interface synchronized with sound in real time.

Performance: We first compare the CPU implementation of manifold filters [1] with our GPU implementation. Since the problem fits well for streaming process, we expect the GPU implementation to outperform the CPU counterpart (see Table I, comparing a single mesh deformation on CPU with 6 and 12 deformations on GPU). Furthermore, we validated that the GPU implementation supports real-time rendering to keep synchronization with the sound. Those experiments allow estimating the appropriate gallery size depending on the graphics hardware (see Table I). We conclude that for models with around 50,000 vertices, a correct gallery size would be between 6 and 12 on a GeForce 130 with 48 cores at 500 MHz.

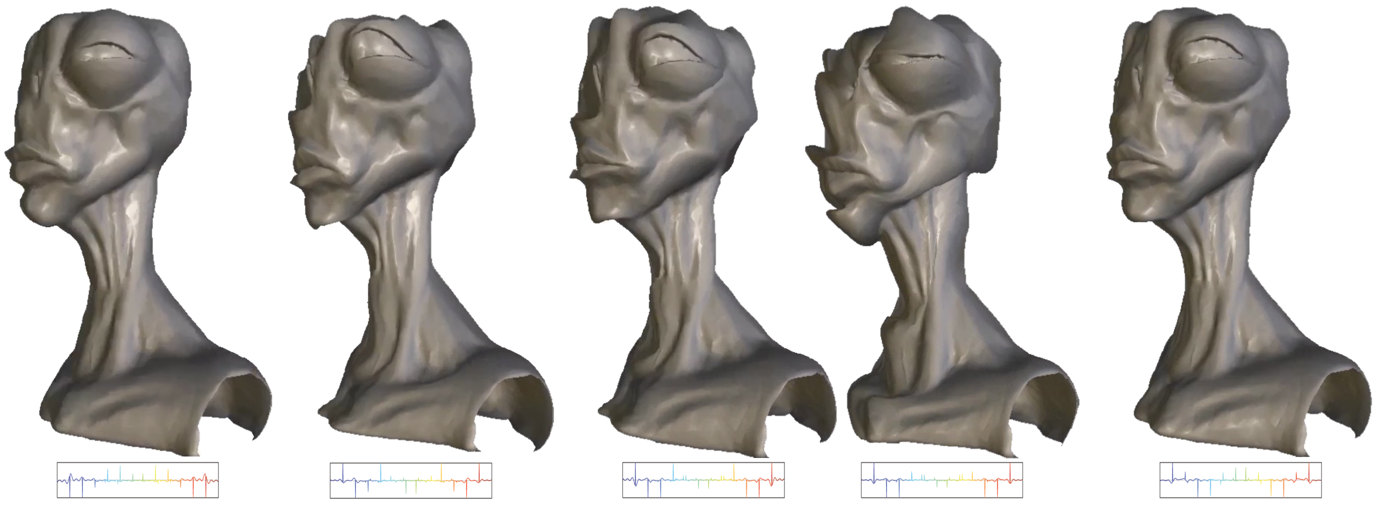


Fig. 9. Frames of electronic music visualization using the alien head model.

TABLE I

Performance tests: all models are normalized into a $[-1,1]^3$ bounding box, and the gallery of S items is rendered in a 1024×768 window. All experiments are performed on a $3.06GHz$ processor with a GeForce GT 130 with $512MB$ of RAM. The deformation speed is measured in frame per second (*fps*), while the harmonic basis pre-computation time is expressed in seconds.

	#verts	**#freqs**	**pre-**	**CPU**	**GPU**	**GPU**
	n	$\#k$	**process**	**S=1**	**S=6**	**S=12**
model			*secs*	*fps*	*fps*	*fps*
pig	1 843	184	4	54.2	203.0	122.8
triceratops	2 832	256	7	46.9	153.9	63.6
neptune	9 392	276	32	12.7	58.6	31.2
multitorus	11 898	270	43	11.5	73.9	40.1
dinosaur	14 054	533	74	4.7	28.3	13.9
octa	15 136	529	25	4.1	26.8	13.2
octopus	20 351	546	129	3.0	21.4	7.1
alien	24 988	540	232	3.4	22.4	12.1
david	24 988	804	116	2.6	14.8	7.3
david head	30 058	1317	232	1.8	15.0	7.0
cat	30 059	271	379	9.6	45.7	23.0
gargoyle	30 059	1052	96	2.1	13.7	6.9
bunny	34 834	1070	478	1.7	12.1	5.8
buste	37 874	1075	329	1.5	13.4	3.4
blooby	42 432	1065	553	1.5	12.5	7.5
egea	63 739	275	718	0.9	7.2	3.2
head	65 002	1607	739	0.5	8.7	4.3
armadilo	86 488	2376	1 134	0.2	10.3	3.4

Music visualization: We use our music visualization for deforming different models in real-time (see the accompanying video). Since the music is decoded and analyzed on the CPU, the combination of sound does not alter the performance of the gallery. We introduce a callback that update the filter every 50 milliseconds, and the rendering is done following the rendering cycles, so that even with large galleries that would harm the real-time rendering, the sound playing does not stop. Finally, we add a parameter $m \in [0,1]$ to control how smoothly the frequencies are passed to the mesh: the sound frequency amplitudes $\tilde{f}(\omega)$ passed to the filter are continuously averaged by $\tilde{f}_{new}(\omega) = w \cdot \tilde{f}_{old}(\omega) + \tilde{f}(\omega)$. For very rhythmic music, this avoids flickering effects on the mesh (see Figs. 1, 6, 9, and 11, and the accompanying video).

Limitations: The GPU implementation allows real-time animated galleries, but it prevents complex processing or further control on the deformed mesh. In particular, it does not permit to directly used quality measure or more advanced interface such as intelligent galleries [24]. The proposed method generates exciting animation on top of a given music. However, we used a very raw sound analysis, which can be enhanced to get more correlated effects. Several complementary effects, in particular on the mesh textures, could improve our music visualization.

VII. Conclusion

In this paper we proposed a GPU implementation of manifold harmonics filters, which allows computing and rendering spectral mesh deformations at a very high rate. We applied this technique for music visualization, using animated design galleries for navigation between different visual effects. Each effect is represented as a mapping from music frequencies to manifold harmonics. We represent such mapping in a concise way to be able to couple genetic reproduction in the gallery.

Acknowledgment

This paper is the result of many discussions between the numerous authors. It has been partially financed by CNPq, FAPERJ and FAPEAL.

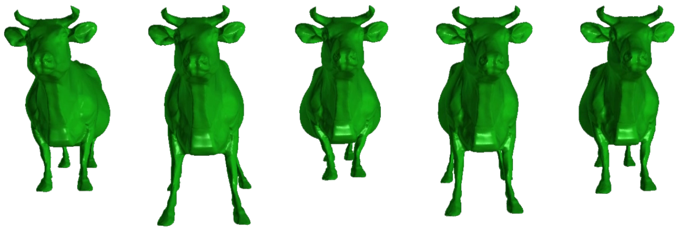


Fig. 10. Frames of reggae music visualization using the cow model.

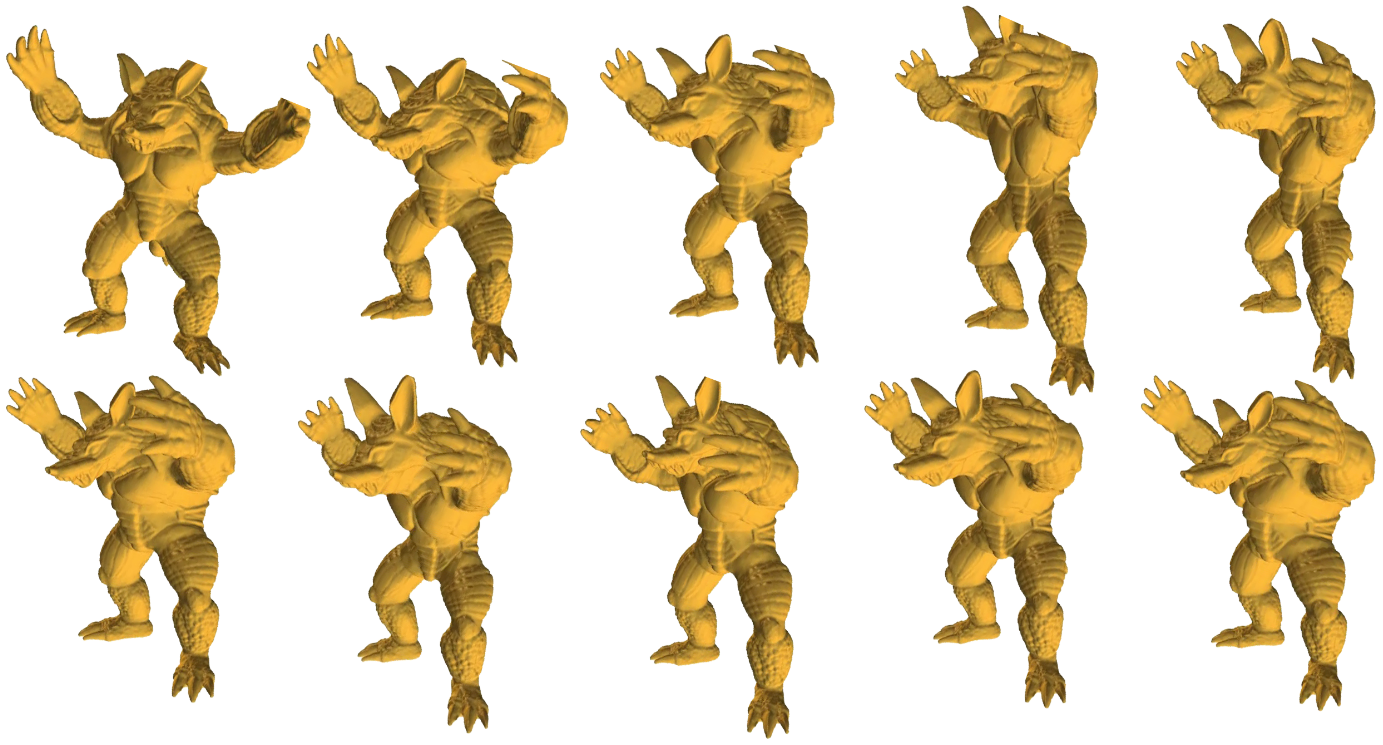


Fig. 11. Frames of rock music visualization using the armadillo model.

References

[1] B. Vallet and B. Lévy, "Spectral geometry processing with manifold harmonics," in *Computer Graphics Forum*, vol. 27, no. 2, 2008, pp. 251–260.

[2] F. de Moura Pinto and C. M. D. S. Freitas, "Two-level interaction transfer function design combining boundary emphasis, manual specification and evolutive generation," in *Sibgrapi*. IEEE, 2006, pp. 281–288.

[3] A. Bordignon, L. Sigaud, G. Tavares, H. Lopes, T. Lewiner, and W. Morgado, "Arch generated shear bands in granular systems," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 11, pp. 2099 – 2108, 2009.

[4] H. Jenny, *Cymatics: A Study of Wave Phenomena & Vibration*, 3rd ed. Macromedia, 2001.

[5] R. W. Clough and J. Penzien, *Dynamics of Structures*. Mcgraw-Hill, 1975.

[6] A. Pentland and J. Williams, "Good vibrations: modal dynamics for graphics and animation," *Siggraph*, vol. 23, no. 3, pp. 207–214, 1989.

[7] C. Yinghui, W. Jing, and L. Xiaohui, "Real-time deformation using modal analysis on graphics hardware," in *Graphite*. ACM, 2006, pp. 173–176.

[8] J. F. O'Brien, C. Shen, and C. M. Gatchalian, "Synthesizing sounds from rigid-body simulations," in *Symposium on Computer animation*. ACM, 2002, pp. 175–181.

[9] G. Taubin, "A signal processing approach to fair surface design," in *Siggraph*, 1995, pp. 351–358.

[10] B. Lévy and H. R. Zhang, "Spectral mesh processing," in *Siggraph Asia Course Note*. ACM, 2009, pp. 1–47.

[11] G. Rong, Y. Cao, and X. Guo, "Spectral mesh deformation," *The Visual Computer*, vol. 24, no. 7, pp. 787–796, 2008.

[12] Y. Liu, B. Prabhakaran, and X. Guo, "A robust spectral approach for blind watermarking of manifold surfaces," in *Multimedia and Security*. ACM, 2008, pp. 43–52.

[13] K. Wang, M. Luo, A. Bors, and F. Denis, "Blind and robust mesh watermarking using manifold harmonics," in *ICIP*. IEEE, 2009, pp. 3657–3660.

[14] M. Ovsjanikov, J. Sun, and L. Guibas, "Global intrinsic symmetries of shapes," in *SGP*. Eurographics, 2008, pp. 1341–1348.

[15] H.-Y. Wu, T. Luo, L. Wang, X.-L. Wang, and H. Zha, "3D shape retrieval by using manifold harmonics analysis with an augmentedly local feature representation," in *VRCAI*. ACM, 2009, pp. 311–313.

[16] J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml *et al.*, "Design galleries: A general approach to setting parameters for computer graphics and animation," in *Siggraph*. ACM, 1997, p. 400.

[17] "The OpenGL Shading Language v 4.0," 2010, www.opengl.org/ documentation/ glsl.

[18] Apple, "PBORenderToVertexArray: render-to-vertex-array using FBO, PBO and VBO," 2006, developer.apple.com/ mac/ library/ samplecode/ PBORenderToVertexArray.

[19] F. Patin, "Beat detection algorithms," 2003, www.gamedev.net/ reference/ programming/ features/ beatdetection.

[20] V. Hernandez, J. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *Transactions on Mathematical Software*, vol. 31, no. 3, p. 362, 2005.

[21] M. Lage, T. Lewiner, H. Lopes, and L. Velho, "CHF: a scalable topological data structure for tetrahedral meshes," in *Sibgrapi*. IEEE, 2005, pp. 349–356.

[22] F. Bellard, "FFmpeg," 2004, www.ffmpeg.org.

[23] G. Hiebert, "OpenAL programmer's guide," 2005, connect.creativelabs.com/ openal.

[24] T. Vieira, A. Bordignon, A. Peixoto, G. Tavares, H. Lopes, L. Velho, and T. Lewiner, "Learning good views through intelligent galleries," *Computer Graphics Forum (Eurographics Proceedings)*, vol. 28, no. 2, pp. 717–726, 2009.