Acta Crystallographica Section A
Foundations of Crystallography
ISSN 0108-7673

Received 7 October 2009
Accepted 4 July 2010

Interactive visualization of quantum-chemistry data

Yun Jang[a]* and Ugo Varetto[b]

[a]Computer Graphics Laboratory, ETH Zurich, Switzerland, and [b]Swiss National Supercomputing Centre, Switzerland. Correspondence e-mail: jangy@inf.ethz.ch

Simulation and computation in chemistry studies have improved as computational power has increased over recent decades. Many types of chemistry simulation results are available, from atomic level bonding to volumetric representations of electron density. However, tools for the visualization of the results from quantum-chemistry computations are still limited to showing atomic bonds and isosurfaces or isocontours corresponding to certain isovalues. In this work, we study the volumetric representations of the results from quantum-chemistry computations, and evaluate and visualize the representations directly on a modern graphics processing unit without resampling the result in grid structures. Our visualization tool handles the direct evaluation of the approximated wavefunctions described as a combination of Gaussian-like primitive basis functions. For visualizations, we use a slice-based volume-rendering technique with a two-dimensional transfer function, volume clipping and illustrative rendering in order to reveal and enhance the quantum-chemistry structure. Since there is no need to resample the volume from the functional representations for the volume rendering, two issues, data transfer and resampling resolution, can be ignored; therefore, it is possible to explore interactively a large amount of different information in the computation results.

1. Introduction

Quantum chemistry applies quantum mechanics and field theory concepts to give a complete and detailed description of the electronic structure of a chemical system. Detailed electronic structure information is key in understanding chemical reactions and, in general, the physical properties of materials. Applications of quantum chemistry include predicting or confirming radiation spectra, studying chemical reactions to understand how two molecules (*e.g.* a possible drug and an enzyme) can interact, molecular design and computational materials science. Quantum chemistry is based on the Schrödinger equation, in which electrons are considered as wave-like particles whose behavior is mathematically represented by a set of wavefunctions obtained by solving the Schrödinger equation that defines the state of a physical system on the atomic scale. The time-independent Schrödinger equation for a one-particle system is

$$\left(\frac{-h^2}{8\pi^2 m}\nabla^2 + \mathbf{V}\right)\psi(\mathbf{r}) = \mathbf{E}\psi(\mathbf{r}), \qquad (1)$$

where h is Planck's constant, m is the mass of the particle, $\mathbf{V}$ is the potential energy, $\mathbf{E}$ is the total energy and $\mathbf{r}$ represents a position in three-dimensional space. The quantity $|\Psi(\mathbf{r})|^2$, where $\Psi(\mathbf{r})$ is a solution to equation (1), gives the probability of finding the particle at position $\mathbf{r}$. Analytical solutions to equation (1) are only available for very simple systems. Quantum-chemistry computations focus on finding approximated solutions to the Schrödinger equation for multi-atom systems. Solutions to equation (1) are used to define properties, represented as scalar fields, such as electron density, electrostatic potential and molecular orbitals.

To analyze properly the output of quantum-chemistry computations, three-dimensional visualizations of atomic orbitals (AOs), molecular orbitals (MOs), electron and spin density, and electrostatic potential are required. Visualization tools currently available in programs like *Molden* (Schaftenaar & Noordik, 2000) or *Molekel* (Molekel, 2009) display scalar fields by first sampling the solutions, which are functional representations, on three-dimensional regular grids and then building isosurfaces using triangulations for specific isovalues. The resampling process prior to the volume rendering of molecular data raises the following challenges: data storage, data transfer and evaluation speed of the functional representations. Since there are multiple atomic and molecular orbitals, it is almost impossible to interactively render all combinations of these atomic and molecular orbitals for atomic and molecular structures. For example, the C_2H_5 molecule is one of the simplest Gaussian-type orbital data sets we studied here and it has 40 different atomic orbitals and 38 different molecular orbitals. Exploring all different combinations is a daunting task because of the $2^{40} \times 38$ resampling processes required. In this work, we present a novel method for performing interactive evaluation and volumetric rendering of atomic and molecular orbitals directly on graphics hardware without resampling. Since the approximate

solutions to the Schrödinger equation are sums of basis functions, we evaluate the functional representations on the fly in a fragment program by storing all parameters and coefficients in textures. We apply general volume rendering, volumetric isosurface rendering, illustrative rendering and volume-clipping techniques in order to visualize the atomic and molecular structures. Fig. 1 gives an overview of our system. We also extend conventional slice-based volume rendering (Wilson *et al.*, 1994).

Our contributions from this work are as follows:

(*a*) Direct evaluation of the functional representation for molecular data on graphics hardware – computing accurate gradients and avoiding data transfers from the central processing unit (CPU) to the graphics processing unit (GPU).

(*b*) Interactive volume rendering of molecular data without resampling on grid structures – discarding an expensive intermediate process for volume rendering.

(*c*) Illustrative and volume-clipping rendering to show nested volumetric atomic and molecular structures – multiple isosurfaces.

In this work, we use two types of basis functions. One is pure Gaussian and the other is a Gaussian-type orbital (GTO). The GTO has been used widely because it is easy to change its shape according to the orbital, whereas the pure Gaussian basis function is newly proposed because of its simplicity in quantum-chemistry computations. However, the technique proposed here can be extended further to perform interactive evaluation and rendering of any functionally represented data set.

We first review previous work in §2 and describe data from quantum-chemistry computations in §3. We then introduce details of our interactive evaluation and visualization of the molecular data and present results produced by our system in §§4 and 5. Finally, the conclusion and future directions are described in §6.

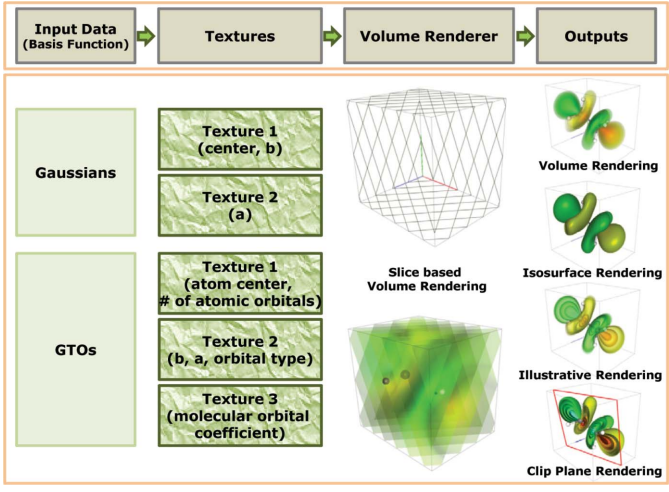


Figure 1
Overview of our interactive visualization system. Two types of data are stored in textures and evaluated and visualized in our volume rendering. Different rendering results, such as volume rendering, isosurface rendering, illustrative rendering and volume-clipping rendering, are produced. Note that *a* and *b* are the coefficients in equation (3).

2. Previous work

In the area of molecular research, there are many visualization studies from drawing simple atoms to rendering volumetric representations. One approach to showing molecular surfaces is to use triangular meshes. Cheng & Shi (2004) proposed the restricted Delaunay triangulation to extract high-quality, smooth, molecular skin surfaces. Another molecular-surface representation was studied by Cipriano & Gleicher (2007), who showed that the abstracted molecular surfaces provide the boundary of a molecule, and that the physical and chemical properties at the boundary are provided by extracting the surface abstract from input triangular meshes. Lampe *et al.* (2007) presented protein dynamics using a two-level rendering approach. They generated geometry residues on the fly to show interactive protein dynamics with 'balls and sticks' for atoms and bonds. When there are many atoms in the visualization, it is difficult to see the overall molecular structure with direct illumination. To enhance visual perception, Tarini *et al.* (2006) presented ambient occlusion and edge cueing.

Volume visualization is also applied to molecular data for volumetric structures. Hu *et al.* (2006) presented direct volume rendering of protein data and they studied an improved transfer function to show various data ranges. In their work, they resampled scalar data to three-dimensional regular grids. Lattice-based volume visualization was presented by Qiao *et al.* (2005). They stored all lattice information from quantum-dot simulations and visualized electron orbitals in a volume. Their work is based on the sampled lattices which are already provided from the computation. Sampled lattices are a discrete form of data, whereas our approach is to handle the continuous data form. Network-based visualization of nanotechnology applications was studied by Qiao *et al.* (2006) and electron particles were visualized with volume rendering of the electron-density volume.

Several researchers present physical and chemical properties of molecular data using visualizations. Lee & Varshney (2002) represented thermal vibrations and uncertainty of the molecular surfaces, and visualized the fuzzy molecular surfaces to provide a more informative display for a better understanding of protein structure and function. Mehta *et al.* (2004) detected anomalous structures in lattice-based molecular-simulation data on a regular grid and showed and verified the detection with visualization. Similar research was presented by Mehta & Jankun-Kelly (2006) on unstructured models of nematic liquid crystals. Another study on cluster detection in molecular dynamics was presented by Grottel *et al.* (2007) with visual verification and analysis. Schmidt-Ehrenberg *et al.* (2002) presented molecular conformations by visualizing regular-grid molecular data.

In order to emphasize molecular structures, chemists use different primitives, such as 'balls and sticks', helices and ribbons. Liu *et al.* (2008) presented interactive molecule construction on a GPU. They reconstructed atoms with spheres interactively with GPU acceleration for educational purposes. Bajaj *et al.* (2004) studied the primitives recon-

Table 1
Polynomial functions (f_t) used in equation (3).

Basis function	Orbital type	f_t
Gaussian	all	1
GTO	s	1
	p_x	x
	p_y	y
	p_z	z
	d_{xx}	$x^2/3^{1/2}$
	d_{yy}	$y^2/3^{1/2}$
	d_{zz}	$z^2/3^{1/2}$
	d_{xy}	xy
	d_{xz}	xz
	d_{yz}	yz

structed on programmable graphics units by three-dimensional image-based rendering. Recently, Stone *et al.* (2009) proposed a fast way to resample the results of quantum-chemistry computations using GPUs and multi-core CPUs. This work was motivated by the heavy computation needed for visualizations of results from quantum-chemistry computations. They used graphics hardware to resample the results on regular grids and visualize the resampled volumetric data. A large increase in speed is achieved compared to a number of different CPU cores and different GPUs. Ufimtsev & Martinez (2008) used a similar approach using GPUs to evaluate the results without rendering capability, whereas our approach combines both the evaluation and the rendering.

Volume rendering is widely used and the techniques vary according to the input grid structures. Jang *et al.* (2004, 2006), however, presented reconstruction of volumetric functional representations using graphics hardware without resampling on grid structures. Ebert *et al.* (2002) showed procedural textures which are continuous and can be evaluated easily in a volume. Since the quantum-chemistry computations generate functional representations of atomic and molecular orbitals, their work motivates our volumetric visualizations of molecular data. Volume illustration is another technique for enhancing the visual understanding of volumetric data and it shows enhanced understanding of medical and flow data (Ebert & Rheingans, 2000; Svakhine *et al.*, 2005).

3. Data in quantum chemistry

Most quantum-chemistry programs find an approximated solution to the Schrödinger equation, then generate data containing a description of the electron structure of the system under analysis in terms of coefficients to be applied to a set of basis functions such as

$$\varphi(x, y, z) = a f_t(x, y, z) \exp(-br^2), \quad (2)$$

where a and b are real values and $f_t(x, y, z)$ is a polynomial function. r is the distance between the center of the basis function and (x, y, z). $f_t(x, y, z)$ varies depending on the orbital types and computation methods. Based on the basis function $\varphi(x, y, z)$, atomic and molecular orbitals are reconstructed as described in the following sections.

3.1. Atomic orbitals

An atomic orbital is a mathematical function that describes the behavior of an electron in an atom. This function can be used to calculate the probability of finding any electron of an atom in a region of space surrounding the atom's nucleus. The term may also refer to the region of three-dimensional space where the electron is *most likely* to be.

Each atomic orbital is approximated with a sum of basis functions of the form described in equation (2),

$$\chi(x, y, z) = \sum_{i=1}^{M} \varphi_i(x, y, z) = \sum_{i=1}^{M} a_i f_t(x, y, z) \exp(-b_i r_i^2), \quad (3)$$

where M is the number of basis functions used to define an orbital and $f_t(x, y, z)$ is a polynomial function defined according to the orbital types (s, sp, p, d, f). The polynomial functions (f_t) for the most common orbitals, s, p and d, are summarized in Table 1.

3.2. Molecular orbitals

Molecular orbitals are mathematical functions which do not represent any physical quantity; they are very useful in the qualitative description of bonding and in studying chemical reactions. The molecular orbital is represented with a linear combination of atomic orbitals,

$$\psi(x, y, z) = \sum_{j=1}^{N} g_j \chi_j(x, y, z), \quad (4)$$

where g_j is a real value (molecular orbital coefficient) and $\chi_j(x, y, z)$ is an atomic orbital introduced in equation (3). N is the number of the atomic orbitals.

3.3. Data structure

As mentioned in §1, we use two different basis functions in this work. In the case of Gaussian basis functions, data are stored as sets of centers (x, y, z), exponents (b) and coefficients (a). Since the Gaussian basis function has one polynomial function, as shown in Table 1, it is not possible to represent different types of atomic orbitals using basis functions with the same center. Therefore, each Gaussian has its own center. Note that there is no molecular orbital coefficient in the Gaussian data format. On the other hand, data represented with GTOs have common centers only at the atom centers, since the orbital types can be represented by the polynomial functions shown in Table 1. The other parameters (b's and a's) and orbital types are stored afterwards, followed by molecular orbital coefficients (g's). The number of molecular orbital coefficients corresponds to the number of atomic orbitals. For example, let us assume that there are two s orbitals, three p orbitals and two d orbitals. The total number of atomic orbitals (*i.e.* the number of molecular orbital coefficients) is $2 \times 1_{(s)} + 3 \times 3_{(p)} + 2 \times 6_{(d)} = 23$, since there are three polynomial functions (f_t) in the p orbital and six polynomial functions in the d orbital.

3.4. Data pre-processing

The coefficients a_i and b_i of atomic orbitals, together with orbital types and the coefficients g_i of the molecular orbitals, are read from the output of popular quantum-chemistry packages such as *Gaussian* and *GAMESS*, and used to reconstruct the molecular orbital functions as described in equation (4). Coefficients are properly normalized to ensure that the probability of finding each electron in the system in the entire three-dimensional space is always equal to 1.

We also find the bond structures among atoms when the data are read and they are sent to the GPU to evaluate 'balls and sticks' for atoms and bonds. Since it is still difficult to find the solution of the Schrödinger equation for molecules with many atoms, we are generally dealing with a small number of atoms. Therefore, we can afford to compute distances for all pairs of atoms in a two-dimensional square array on the CPU and compare the distance with the covalent radius (Allen *et al.*, 1987). If the distance between two atoms is less than the covalent radius, there is a bond between them.

4. Interactive visualization of molecular data

As described in §3, the molecular data in quantum chemistry are modeled as a sum of basis functions, such as Gaussians and GTOs. The molecular data file contains sequences of basis-function parameters including centers, coefficients, exponents *etc*. Some basis functions, such as GTOs, imply different reconstruction equations depending on the orbital types. The reconstruction (resampling) of data values in a certain volume would require time-consuming computations proportional to the selected grid resolutions and the visualization of the reconstructed volume would incur a high level of computational pre-processing with conventional direct volume-rendering techniques. Seeing the details of molecular structures requires many combinations of orbitals, which makes it impossible to pre-compute and store the resampled volumes.

Avoiding completely this resampling approach used for most of molecular visualization research, we reconstruct the volume directly on the GPU by storing the basis-function parameters in textures. With our approach, it is possible to show the volumetric molecular structures without transferring massive amounts of volumetric data.

4.1. Volume rendering on a GPU

Volume rendering (Cabral *et al.*, 1994) in visualization is a common technique and there are many approaches depending on underlying grid structures. In our molecular visualization, the data do not lie on a specific grid structure. We therefore chose the slice-based volume-rendering technique and we evaluate fragments on each slice in a fragment program. The slices are rendered from back to front, so that the color and opacity are properly displayed. The slices are generated by computing intersections between a bounding box and planes in a vertex program as proposed by Rezk-Salama & Kolb (2005). The number of slices can be adjusted by our user interface. Using this volume-rendering base, a two-dimensional transfer function is used to explore the interesting data values in the volume.

Currently, we use two different types of molecular data from quantum-chemistry computations. One is formed with all Gaussians and the other is formed with GTOs. The Gaussian basis functions have centers, exponents and coefficients of Gaussians as parameters. On the other hand, GTOs have centers of atoms, exponents and coefficients of atomic orbitals, and coefficients of molecular orbitals. In this work, we focus on two different types of orbitals, namely atomic and molecular orbitals. For the atomic orbitals, only basis-function parameters are needed for the reconstruction, whereas the molecular orbitals require one additional parameter, which is the molecular orbital coefficient.

4.2. Texture layouts for molecular data

We store all these basis-function parameters and corresponding coefficients in two-dimensional textures (texture memory on the GPU) and fetch the texture values in a fragment program for evaluating data values and gradients of the functional representations. In order to evaluate the functional representation efficiently in a fragment program, we encode all parameters and coefficients as shown in Fig. 2. Since we use two types of basis functions, we introduce two different texture layouts, one for Gaussians and the other for GTOs.

The top layout in Fig. 2 shows how we encode the texture for Gaussian basis functions. Since a Gaussian basis function does not imply different polynomial functions according to atomic orbital types, we simply store the centers (x_i, y_i, z_i) and exponents b_i in one texture, and coefficients a_i in another texture. Note that each texture memory contains four components (r, g, b, a). Therefore, simple fetching of the texture values in a fragment program is possible in order to obtain the sum of all basis functions. Note that there is no molecular orbital coefficient in our molecular data.

In the evaluation of GTOs shown in equation (3) there is a polynomial term $f_t(x, y, z)$, which is defined according to the atomic orbital type. There is only one common center with different atomic orbital parameters for the different atomic orbital types in one atom. The shape of an orbital is defined by the polynomial term and the polynomial terms can be multiple functions for p, sp, d and f. For example, the p orbital has three polynomial terms (p_x, p_y, p_z) and the d orbital has six polynomial terms $(d_{xx}, d_{yy}, d_{zz}, d_{xy}, d_{xz}, d_{yz})$. Each polynomial term has its own molecular orbital coefficient. The bottom part of Fig. 2 shows the texture layouts for the atomic orbitals and molecular orbitals with two GTO basis functions. In this example, there are two atoms, nine atomic orbital types with 16 sets of parameters, and two sets of molecular orbital coefficients for 36 atomic orbitals. There are multiple basis functions for one orbital type. In this example, the first s orbital ($s1$) for atom 1 has two sets of parameters and the first d orbital for atom 2 ($d1$) has three sets of parameters. The number 36 is calculated as $1_{(s1)} + 3_{(p1)} + 3_{(p2)} + 6_{(d1)} + 10_{(f1)} + 1_{(s1)} + 3_{(p1)} + 3_{(p2)} + 6_{(d1)}$. As shown in the figure, we put the atom centers and the number of orbital types in texture 1.

Texture 2 is composed of exponents, coefficients and orbital types for atomic orbitals. Then we store the molecular orbital coefficients in texture 3. Textures 1 and 2 are organized in the order of the parameters, whereas texture 3 is designed according to the orbital type. For example, since the *s* orbital has only one polynomial term, there is one molecular orbital coefficient, which is stored in r out of rgba. The *p* orbital has three polynomial terms; therefore, we place three molecular orbital coefficients in one rgb in order to reduce the texture fetch. In the same way, for the *d* orbital, we use two rgb's for the six molecular orbital coefficients.

4.3. Per-fragment reconstruction

In order to evaluate fragments, we use a high-level language, NVIDIA Cg (NVIDIA, 2009), and the Cg code is compiled on the fly after the molecular data are read. With the support of Cg language, we use an `if` statement to choose the appropriate fragment program for the functional evaluations depending on the basis functions. Since we have two different basis functions, we show two different fragment programs according to the basis functions. The functional values and the gradients are evaluated at the same time.

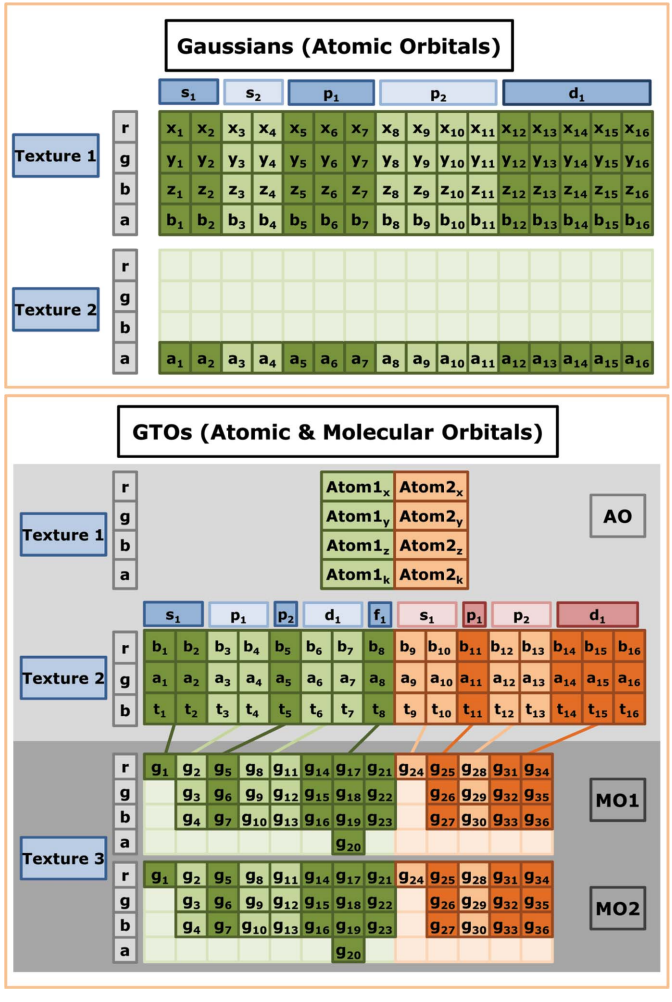


Figure 2
The parameters of basis functions are packed into two or three texture memories. Each texture memory contains four components rgba. The top image shows an example of texture layouts for a 16 Gaussian basis function with five atomic orbitals in two texture memories. The bottom image presents our texture packing of GTOs into three textures. In this example, there are two atoms, nine atomic orbital types, 36 atomic orbitals and two sets of the molecular orbital coefficients.

For Gaussian basis functions, two textures, as shown in the top part of Fig. 2, are fetched for all parameters including the centers of basis functions and the atomic orbitals are evaluated using equation (3). Fig. 3 presents a pseudo-Cg code for the reconstruction of Gaussian basis functions. In the Cg code, `ithOrbitalDraw` is connected to our user interface; therefore, we can select/deselect any orbitals on the fly.

For the GTO basis functions, each fragment is evaluated by computing either equation (3) for the atomic orbitals or equation (4) for the molecular orbitals. The atomic orbitals are computed by two texture lookups (texture 1 and texture 2 from the GTOs texture layout in Fig. 2). Note that the texture lookup is performed with `texRECT` in Fig. 3. We fetch texture 1 for the atom centers and the number of atomic orbital types. Then we fetch texture 2 for the exponents, coefficients and atomic orbital types of each atomic orbital. Once we have all the parameters, equation (3) is evaluated with the polynomial functions shown in Table 1 in the `for` loop. In the fragment program, we can handle all four rgba (= *xyzw*) components in parallel, for example, `inpos.xyz` and `texValue1.xyz`. In the example, we do not need to use the 'a' component. Note that `texValue1` has all four components (*xyzw*) from the `Texture1`.

On the other hand, the molecular orbital evaluation needs one more computation based on the atomic orbital evaluation as described in the previous paragraph. We fetch one more texture (texture 3) for the molecular orbital coefficients and multiply the coefficients right after the atomic orbital evaluation. Fig. 4 shows a pseudo-Cg fragment program for this molecular orbital evaluation. Specifically, we show the functional value and gradient calculation with a *p* orbital in order to show the efficiency of our texture layout (texture 3 from the GTO texture layout in Fig. 2). As shown in the Cg code, we can evaluate three different *p* orbitals (p_x, p_y, p_z) at the same time by fetching three molecular coefficients by one

```
for(i = 0; i < NumOfOrbitalTypes; i++){
 if(ithOrbitalDraw){
  for(j = ithOrbitalStart; j < ithOrbitalEnd; j++){
    // for x,y,z, and exponent
    texValue1 = texRECT(Texture1, texpos(j));
    // for coefficient
    texValue2 = texRECT(Texture2, texpos(j));
    r = inpos.xyz - texValue1.xyz;
    // Data value
    f = texValue2.w *
        exp((-1.0) * texValue1.w * dot(r, r));
    // Gradient
    df = (-2.0) * texValue1.w * f * r;
    val += float4(df, f);
}}}
```

Figure 3
The main loop of a pseudo-Cg fragment program for the evaluation of an atomic orbital with Gaussian basis functions.

texture lookup. Similar computation is applied to *d* and *f* orbitals.

Once the value and gradient of a fragment are computed, we fetch the transfer-function texture for color and opacity. Then we apply illumination, illustrative rendering techniques and a volume-clipping technique in the fragment program. Note that the gradient is also used as a normal vector at a position in the volume. Therefore, it is used for the illuminations and illustrative renderings.

```
n = 0;
val = float4(0.0, 0.0, 0.0, 0.0);

for(i = 0; i < NumAtoms; i++)
{
 // for x,y,z and number of orbital types
 texValue1 = texRECT(Texture1, texpos(i));
 r = inpos.xyz - texValue1.xyz;
 for(j = start(i); j < start(i)+texValue1.w; j++)
 {
  for(k = jthOrbitalStart; k < jthOrbitalEnd; k++)
  {
   // for exponent, coefficient, orbital type
   texValue2 = texRECT(Texture2, texpos(k));

   // for s orbital
   if(texValue2.z == 1){
    if(k == jthOrbitalStart){
     // for 1 molecular orbital coefficient
     texValue3 = texRECT(Texture3, texpos(n));
     // increase offset for Texture 3
     n += 1;
    }
    ......
   }
   // for p orbital
   else if(texValue2.z == 3)
   {
    if(k == jthOrbitalStart)
    {
     // for 3 molecular orbital coefficients
     texValue3 = texRECT(Texture3, texpos(n));
     // increase offset for Texture 3
     n += 1;
    }
    tmp1 = dot(r, texValue3.xyz);
    tmp2 =  texValue2.y *
        exp((-1.0) * texValue2.x * dot(r, r));
    // Data value
    f = tmp1 * tmp2;
    // Gradient
    df = texValue3.xyz * tmp2 -
         2.0 * texValue2.x * r * f;
    val += float4(df, f);
   }
   // for d orbital
   else if(texValue2.z == 6){
    if(k == jthOrbitalStart){
     // for 6 molecular orbital coefficients
     texValue3 = texRECT(Texture3, texpos(n));
     texValue4 = texRECT(Texture3, texpos(n+1));
     // increase offset for Texture 3
     n += 2;
    }
    ......
   }
}}}
```

Figure 4
The main loop of a pseudo-Cg fragment program for the evaluation of a molecular orbital with GTO basis functions. In this code, only *p*-type molecular orbital computation is shown because it shows the efficiency of our texture layout for the molecular orbital coefficients. Other orbitals can be evaluated by similar computations.

4.4. Ball-and-stick rendering

Most molecular visualizations provide a drawing of the atoms and bonds between atoms and common primitives for drawing these are balls and sticks. Many molecular visualization tools represent balls and sticks with triangular meshes and render the meshes with other features, such as isosurfaces. In order to render the meshed balls and sticks together with the volumetric representation, however, a visibility test with sorting is necessary for proper rendering results with a single rendering pass. In this work, we avoid the visibility test by evaluating the balls and sticks as functional forms. Balls are represented as spheres and sticks are represented as cylinders. The sphere and cylinder equations are evaluated as solid volumes prior to the evaluation of the functional representations. If a fragment is inside either a sphere or a cylinder, then we avoid the expensive evaluation of the functional representation. The bond structures are pre-processed when the molecular data are read as described in §3.4.

The functional form of the ball is defined as

$$f(x, y, z) = (x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2, \quad (5)$$

where (c_x, c_y, c_z) is an atom center and r is the radius of the ball. The cylinder representation of the stick is defined as follows. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be centers of two atoms and $\mathbf{n}$ be $\mathbf{c}_2 - \mathbf{c}_1$. Also, let (x_1, y_1, z_1) be a point in an arbitrary volume, which is known. Now, we want to find a point (x_0, y_0, z_0) along the line between c_1 and c_2. On the unknown point (x_0, y_0, z_0), we set two primitives, a line parallel and a plane perpendicular to $\mathbf{n}$. The line is defined as

$$\frac{x - c_{1_x}}{n_x} = \frac{y - c_{1_y}}{n_y} = \frac{z - c_{1_z}}{n_z} = t \quad (6)$$

and the plane is defined as

$$n_x(x - x_1) + n_y(y - y_1) + n_z(z - z_1) = 0. \quad (7)$$

We can solve these two equations to obtain (x_0, y_0, z_0), which is an intersection between the line and the plane. Then we compute the distance between (x_0, y_0, z_0) and (x_1, y_1, z_1). If the distance is less than the cylinder radius, the point (x_1, y_1, z_1) is inside the cylinder. Otherwise, it is not. Note that we need to make sure that the intersection point (x_0, y_0, z_0) is between two atom centers.

The ball radius is the van der Waals radius and is read from an atom element table (BlueObelisk, 2008). The radius varies depending on the atomic number. The stick radius is set to half the van der Waals radius of hydrogen, which is the smallest in the table. We also provide a control for both radii in our user interface so that users can change the relative sizes. The atom color is also read from the atom element table, so that chemists understand the molecular structures easily. The functional representations of the balls and sticks are evaluated

as a solid volume in the fragment program before evaluating the molecular data. If a fragment is inside either the sphere or the cylinder, there is no evaluation of the molecular data since the molecular data are hidden.

4.5. Transfer function and illustrative rendering

A transfer function is used to filter data interactively on the graphical user interface. Mostly we provide a one-dimensional or two-dimensional histogram of data and users control the data values or data gradients that are shown in the visualization. In this work, we use a two-dimensional transfer function with data value *versus* its gradient magnitude. For this volume rendering of quantum-chemistry data, multiple isovalues are preferable since more isovalues show the detail of the molecular structures. However, the higher absolute isovalues are nested in the lower absolute isovalues in the molecular data. In order to show the internal structures of a molecule clearly, we design five different two-dimensional transfer functions (TFs): uniform, Gaussian, left half of Gaussian, right half of Gaussian and sinusoidal, as shown in Fig. 5(*a*). The two-dimensional transfer function is a two-dimensional histogram of data values (*x* axis) and gradients (*y* axis). The uniform TF is appropriate for rendering isosurfaces and the Gaussian TF is used for the volume rendering of the molecules. Specifically, the right and left halves of the Gaussian TF are preferable for seeing the nested orbital structures, since the highest or lowest values, which are found mostly at the cores of atoms, are inside outer shells. The right half of the Gaussian TF is used for the negative values and the left half of the Gaussian TF is used for the positive values. Figs. 5(*b*) and (*c*) present the transfer-function comparison of the right half of the Gaussian TF and the whole Gaussian TF with the GTO basis function data set for C_2H_6. The right half of the Gaussian TF shows the core of the orbital, whereas the whole Gaussian TF hides the internal structures.

We also apply illustrative rendering techniques to the molecular data visualization, such as enhancing boundaries with the sinusoidal TF proposed by Ebert & Rheingans (2000) and Svakhine *et al.* (2005). Note that we refer to the works of Ebert & Rheingans (2000) and Svakhine *et al.* (2005) for details. The sinusoidal TF is used to show multiple isovalues (*e.g.* the contour volume) in order to see the atomic and molecular orbital structures. Since visualizing more isovalues at the same time is preferable, the sinusoidal TF with boundary enhancements is used to show the contour volume in an illustrative way, so that we can provide more isovalues and structures in the volume. Fig. 6 shows the rendering results of the 27th molecular orbital of the GTO basis function data for $C_4H_2CH_2CH_2CH_4$. In particular, the boundary enhancement with the sinusoidal TF produces very clear multiple isovalue structures of the nested molecular data.

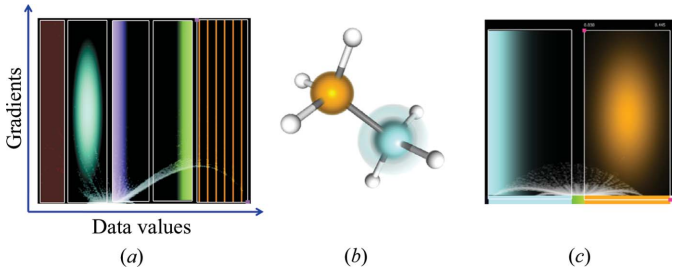


Figure 5
Transfer-function (TF) comparison. (*a*) Our five different transfer functions (from the left: uniform, Gaussian, the right half of Gaussian, the left half of Gaussian and sinusoidal). Note that the *x* axis represents data values and the *y* axis represents gradient magnitudes. (*c*) Our two-dimensional TF setting for generating the image in (*b*). The right half of the Gaussian TF (sky blue) is compared with the simple Gaussian TF (orange). It is possible to see the nested orbital structures in the core of the atom with the right half of the Gaussian TF.

4.6. Volume clipping

Volume clipping is a technique for hiding unimportant parts in the volume rendering. In §4.5, we mention that visualizing multiple isovalues is preferable for showing the molecular structures. Fig. 7(*a*) shows our volume-clipping rendering with Gaussian basis function data for BeO. The atomic orbital structure in the middle of the volume is clearly shown by clipping half of the volume out. Users can change the position and direction of the clipping plane to place it across the area they are interested in. We decide whether a subvolume is visible or not by evaluating the clipping-plane equation and the subvolume location in our fragment program. Then we use the `clip` function from the Cg library to remove unwanted subvolumes.

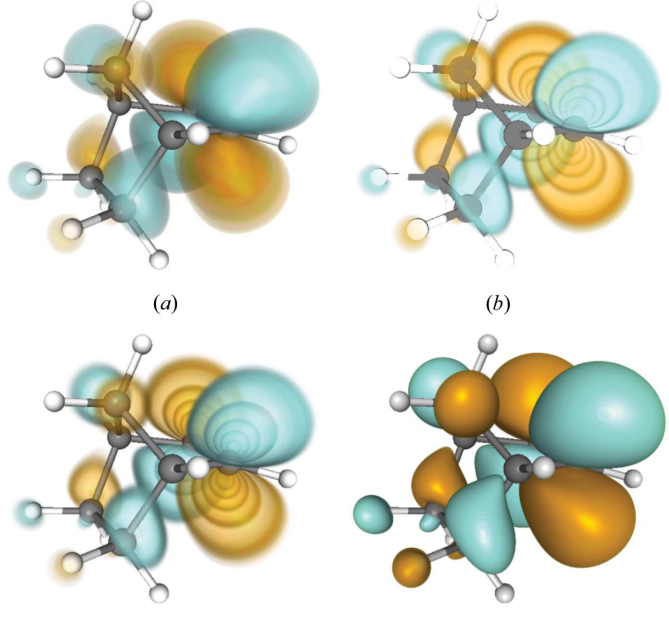


Figure 6
Volume rendering of the 27th molecular orbital for $C_4H_2CH_2CH_2CH_4$. Warm colors represent positive values and cool colors represent negative values [equation (4)]. (*a*) A conventional volume rendering with local illumination. (*b*) A volume rendering with the edge coloring using the sinusoidal TF without illumination. (*c*) A boundary-enhanced contour volume rendering with the sinusoidal TFs. (*d*) An isosurface rendering with local illumination.

4.7. User interface

In our system, there are four main parts of the user interface, including data loader, colormap loader, transfer-function control and rendering control. The data loader and colormap loader parts are simple file-loading interfaces. In the transfer-function control, we designed the user interface to control multiple transfer functions with five different transfer-function modes, which are described in §4.5. Data values, such as minima and maxima, can also be adjusted for various data ranges over many atomic and molecular orbitals in the same data set. In the rendering control, we can select lighting modes, illustrative rendering mode, specific atomic orbitals and molecular orbitals, and control the clipping plane for the volume clipping. In particular, the atomic orbital and molecular orbital selection interfaces can be used to explore the various atomic and molecular orbitals interactively. Fig. 8 shows our atomic and molecular orbital interfaces. Users can select any of the atomic orbitals (Fig. 8*a*) and can also choose any of the molecular orbitals based on the information shown in Fig. 8(*b*).

5. Results and discussion

We implemented our system on a Core 2 Quad CPU 2.4 GHz processor with NVIDIA GeForce GTX 260 graphics hardware. We extended our slice-based volume-rendering system by evaluating the functional values directly on a GPU. Ray casting (Hadwiger *et al.*, 2005; Krüger & Westermann, 2003; Röttger *et al.*, 2003) could also be used for this application and easily integrated into our system, since ray casting has advantages such as adaptive sampling. One issue in our slice-based volume rendering is the number of slices needed to reveal all the properties of a quantum-chemistry study. Resampling with very high resolution takes up to several hours and it is difficult to visualize high-resolution data on a desktop PC. In this sense, changing the number of slices in our system is much easier and faster than resampling and transferring the resampled data.

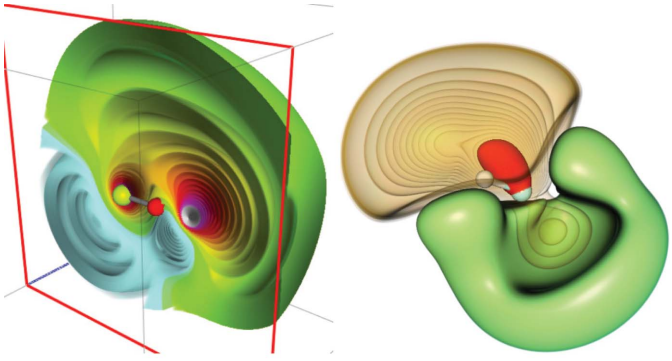
Table 2
Three data sets with Gaussian basis functions.

Data set	No. of basis functions (x, y, z, b, a)	No. of atomic orbital types	No. of atomic orbitals
LiH	20	4	6
HF	66	9	19
BeO	112	12	28

We have tested the various data sets summarized in Tables 2 and 3, where we specify the number of basis functions with the number of corresponding parameters.

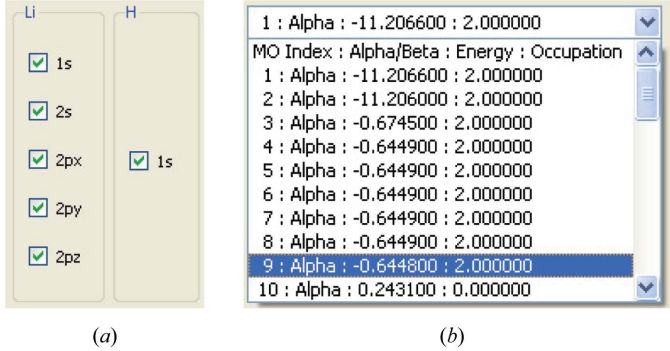
BeO, HF and LiH are data sets using Gaussian basis functions. BeO is presented in Fig. 7(*a*) with multiple isosurfaces and volume clipping. Clipping along the bond between two atoms shows the internal orbital structures of BeO. Fig. 7(*b*) is a rendering result for HF. In this image, some of the positive orbitals are nested in the negative orbitals. Figs. 9(*a*) and (*b*) are progressive atomic orbitals of LiH with boundary-enhanced volume contours. Fig. 9(*a*) shows a combination of 1*s*, 2*s* and $2p_x$ of Li, whereas Fig. 9(*b*) is a rendering result of $2p_y$ of Li, and 1*s* of H on top of part (*a*). Comparing the two images, we can see that the 1*s* of the H orbital (left core) and $2p_y$ of the Li orbital (rotation of negative values) change the atomic orbital structure.

C_2H_5, C_2H_6, $C_4H_2CH_2CH_2CH_4$ and $N_2C_4O_2H_4N_2C_4O_2H_4$ have GTOs as the basis functions. C_2H_5 has 38 molecular orbitals and Fig. 10 shows the third and 21st molecular orbitals with contour volumes. The energy level of the third molecular orbital is −0.7074 with an occupation of two and α spin, whereas the energy level of the 21st molecular orbital is 0.9449 with zero occupation and α spin. C_2H_6 data are used in Fig. 11 compared with the results of *Molekel*. Figs. 11(*a*) and (*b*) of the figure are isosurface rendering of isovalues (±0.05) with different grid resolutions. The grid resolution of (*a*) is 31 × 24 × 24 and that of (*b*) is 170 × 128 × 137. The computation timings for Figs. 11(*a*) and (*b*) in *Molekel* are 0.22 and 32.49 s. As seen in Fig. 11(*a*), the low-resolution isosurface shows artifacts on the surface due to the low resolution of resam-



(*a*) (*b*)

Figure 7
(*a*) Contour volume rendering of the atomic orbitals for BeO (Be is the green atom) with the volume-clipping technique. (*b*) A boundary-enhanced illustrative rendering (positive values, orange) with an isosurface rendering (negative value, green) for HF (H is the white atom). The highest positive value is rendered with the red isosurface.



(*a*) (*b*)

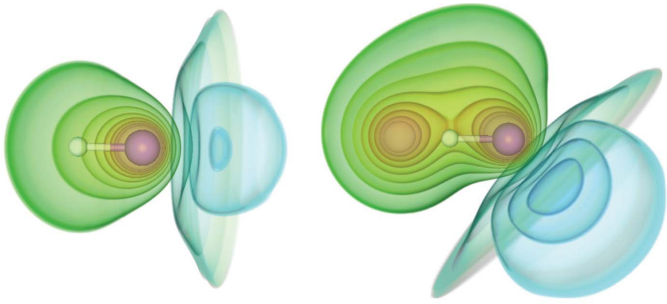
Figure 8
User interfaces for the atomic and molecular orbitals. (*a*) Our user interface for choosing the atomic orbitals. (*b*) The interface for the molecular orbitals. This shows the spin, energy and occupation of each molecular orbital in order to help users to select interesting molecular orbitals.

Table 3
Four data sets with GTO basis functions.

Data set	No. of atoms (x, y, z)	No. of atomic orbital types	No. of basis functions (b, a)	No. of molecular orbital coefficients (g)	No. of molecular orbitals
C_2H_5	7	27	50	40	38
C_2H_6	8	30	54	42	40
$C_4H_2CH_2CH_2CH_4$	17	62	124	125	118
$N_2C_4O_2H_4N_2C_4O_2H_4$	24	120	856	264	162

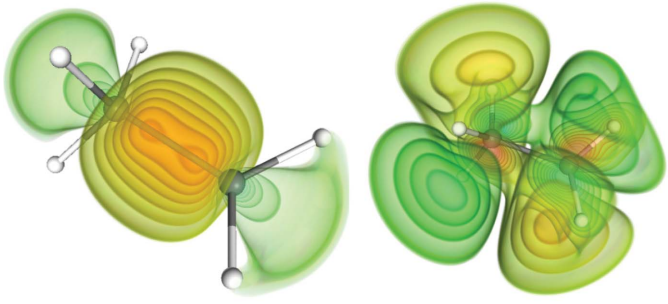
pling. Fig. 11(c) is the isosurface rendering using our system. Our system does not produce any artifacts since we perform per-fragment evaluation of the functional representations. Fig. 6 presents different rendering techniques on $C_4H_2CH_2$-CH_2CH_4 and Fig. 12 presents two different styles for the 30th molecular orbital of $C_4H_2CH_2CH_2CH_4$. Fig. 12(a) shows the positive data values as the contour volume (orange) and the negative values as solid volume (blue). Part (b) is the opposite of part (a). Figs. 13(a) and (b) show different molecular orbitals of C_2H_6. Fig. 13(a) is rendered with the volume-clipping technique and multiple isosurfaces and Fig. 13(b) is generated by the volume contours with the boundary enhancement. Both images present the orbital structures among the atoms and bonds. Fig. 13(c) shows the volume clipping and (d) shows the isosurface with the volume contours of $N_2C_4O_2H_4N_2C_4O_2H_4$. The molecule is very complicated but both images show the molecular orbital structures in the volume.

We also measured performances on a viewport of 600×517 with 256 slices. The performances and storage of the data sets are summarized in Table 4. Data sets with GTO basis functions are compared with performances using *Molekel*. The performances for our system indicate the evaluation and rendering speed, whereas the performances for *Molekel* include only the resampling and triangulation for one isosurface. Note that *Molekel* does not support the data format for Gaussian basis functions and $N_2C_4O_2H_4N_2C_4O_2H_4$ is not readable in *Molekel*. The grid resolution in *Molekel* for the comparison is set as $122 \times 97 \times 97$ and *Molekel* performs resampling in the grid and marching cube for the isosurfaces. As seen in the table, our system provides greater performance than *Molekel*, even though we do not include the rendering speeds in *Molekel*. The performance of our system is highly dependent on the number of basis functions and the atomic orbital types. In particular, d orbitals require more computa-



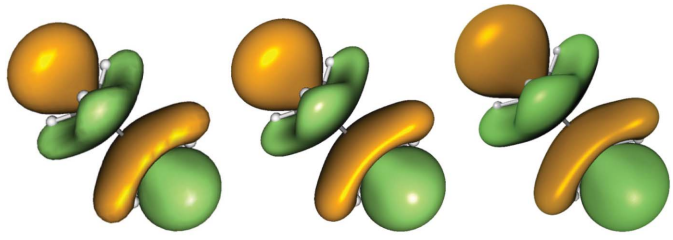
(a) (b)

Figure 9
Renderings of the atomic orbitals for an LiH molecule (H is the white atom) with boundary-enhanced volume contours. (a) The atomic orbitals rendered with only 1s, 2s and 2p_x of Li. (b) The atomic orbitals rendered with 2p_y of Li, and 1s of H on top of (a).



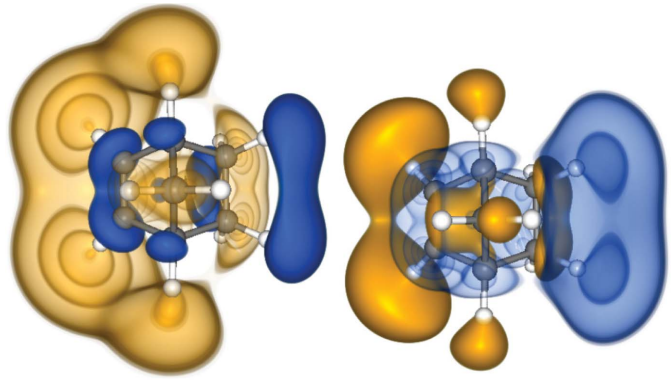
(a) (b)

Figure 10
Molecular orbitals for C_2H_5 (H is the white atom). (a) The third molecular orbital. (b) The 21st molecular orbital.



(a) (b) (c)

Figure 11
Two-isosurface [0.5 (orange) and −0.5 (green)] comparison of *Molekel* (Molekel, 2009) and our system. Parts (a) and (b) show low ($31 \times 24 \times 24$) and high ($170 \times 128 \times 137$) resolutions of the isosurface using *Molekel*. (c) The isosurface rendering using our system.



(a) (b)

Figure 12
Molecular orbitals for $C_4H_2CH_2CH_2CH_4$ (H is the white atom). (a) The 30th molecular orbital with the positive values as the contour volume. (b) The 30th molecular orbital with the negative values as the contour volume.

Table 4
Comparison of performances and data-storage requirements between our system and *Molekel*.

The evaluation and rendering performances for our system are measured on a viewport of 600 × 517 with 256 slices whereas performances for *Molekel* are measured on a regular grid 122 × 97 × 97 only for resampling and triangulation of one isosurface. The storage values indicate the amount of data to be transfered to the GPU.

	Our system		*Molekel*	
Data set	Speed (s)	Storage (kb)	Speed (s)	Storage (kb)
LiH	0.047	0.4	n/a	n/a
HF	0.094	1.3	n/a	n/a
BeO	0.204	2.2	n/a	n/a
C_2H_5	0.108	6.6	11.1	1147.9
C_2H_6	0.140	7.2	12.7	1147.9
$C_4H_2CH_2CH_2CH_4$	0.270	60.2	27.2	1147.9
$N_2C_4O_2H_4N_2C_4O_2H_4$	1.000	178.2	n/a	n/a

tion compared to *s* or *p* orbitals. The storage data in Table 4 indicate the amount of data which need to be transferred to the GPU. Since we evaluate data values on the fly, only parameters and coefficients of the basis functions are required. The storage requirement in *Molekel*, however, is the size of the resampled data, which could be used in generic volume renderers to generate images similar to our system.

Our system has enabled scientists to analyze interactively atomic and molecular orbitals that give a clear idea of intra-molecular bonding properties such as the sites where bonds are more likely to form (bonding sites) and the sites where bonds are not likely to form (antibonding sites). This is achieved by applying a sinusoidal transfer function which results in the rendering of multilevel (*i.e.* multiple isovalues) colored surfaces where the intensity of the color can be set to match the probability of finding electrons at a specific point in space. In particular, Fig. 10(*a*) presents clearly the probability of finding electrons. There is a strong bond between two carbon atoms and there is a weak bond between two hydrogen atoms on the right side, which is not found on the left side of the molecule. The peanut shape of the strong bond was difficult to visualize due to the lack of interactivity in generating many isosurfaces. Moreover, the weak bond is not clearly shown without multiple isovalues. Figs. 13(*a*) and 13(*b*) show how different two molecular orbitals are, according to the energy. Our approach also proves useful for divulgation purposes by giving an intuitive idea of how atoms might bond or not bond to each other by simply looking at the intensity of the colors. Note that this visualization can only be achieved with proper support for transparency when one has triangulated meshes, since the sites where bonds are created are found in the innermost part of the orbital volumes. It is therefore important to show lighter outer areas and inner darker volumes of space at the same time. When applying mesh-based techniques multiple passes are required for correct rendering of overlapping transparent surfaces. Another issue that scientists encounter is the performance of the analysis tool. In order to perform the required analysis, scientists must be able to generate quickly a number of molecular orbitals while varying parameters such as isovalues, bounding box, color and transparency, to help in understanding inter- and intramolecular bonding properties; this is simply impossible with the currently available tools, which may take hours (compared to tenths of a second with our approach) to generate the entire set of orbitals.

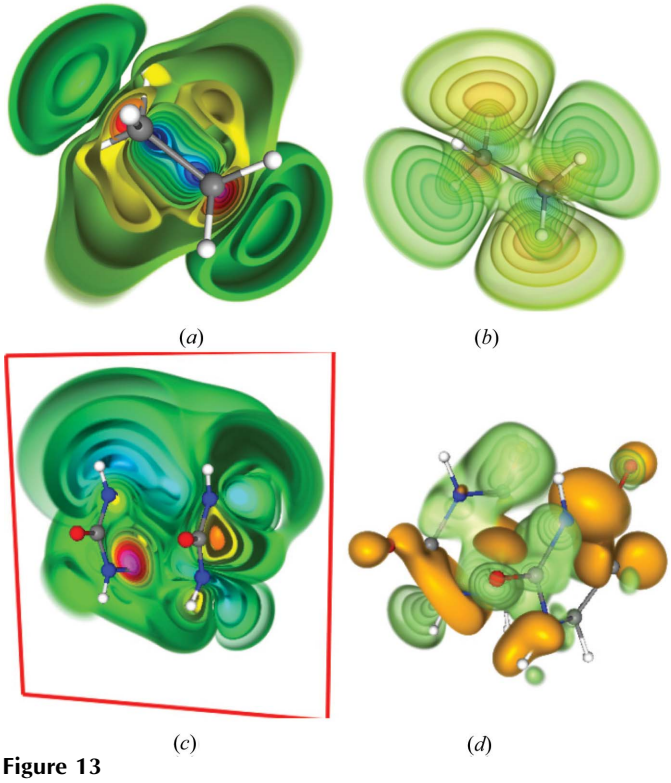


Figure 13
More rendering results for C_2H_6 (*a*, *b*) and $N_2C_4O_2H_4N_2C_4O_2H_4$ (*c*, *d*). Parts (*a*) and (*b*) are the 17th and 20th molecular orbitals with the boundary-enhanced contour volume. (*c*) The isosurface rendering of the 27th molecular orbital with multiple isosurfaces and volume clipping. (*d*) Both isosurfaces (orange) and volume contours with the sinusoidal TF (green) of the 30th molecular orbital.

6. Conclusion and future directions

We have presented our interactive volume rendering of molecular data by evaluating the functional representations on a GPU. Our system does not require the resampling on grid structures for the volume rendering; therefore, there is no data-transfer issue for a high grid resolution. Direct per-fragment evaluation of the functional representation in our fragment program allows us to interactively explore the functional representations of molecular data and generate images without artifacts, which are often seen in grid structures. We also use illustrative rendering techniques to show the nested atomic and molecular structures and our user interface enables us to select any interesting atomic and molecular orbitals.

As seen in Table 4, the performance is degraded as the number of basis functions and parameters increases. A possible solution is to use hierarchical spatial structures as proposed by Jang *et al.* (2004). We will investigate the spatial data structures of the molecular data and we would also like to

compare the evaluation quality and performance with ray casting as an extension of the spatial structure study in the future. Another future work is related to computing electron density and electrostatic potential. In this work, the atomic and molecular orbitals are computed on the fly. The computation of electron density and electrostatic potential, however, requires many redundant processes. In order to improve the performance for electron density and electrostatic potential, it could be better to use multilevel rendering, so that we can reduce the large amount of redundant computations. Moreover, we would like to study our ball-and-stick rendering with this multilevel rendering because the computation of our visibility test is expensive when we have more atoms and bonds, and investigate the surface rendering proposed by Loop & Blinn (2006). Since we can evaluate the scalars and gradients in our fragment program, we are also interested in interactively exploring vector fields in the molecular data, such as topological structures of orbitals and the directions of orbital gradient fields.

The authors would like to thank Jean Favre, Mario Valle, John Biddiscombe and Maria Grazia Giuffreda. This work was supported in part by the Swiss National Science Foundation under grant No. 200021_124642.

References

Allen, F. H., Kennard, O., Watson, D. G., Brammer, L., Orpen, A. G. & Taylor, R. (1987). *J. Chem. Soc. Perkin Trans. 2* pp. S1–S19.

Bajaj, C. L., Djeu, P., Siddavanahalli, V. & Thane, A. (2004). *Proceedings of the IEEE Conference on Visualization 2004*, pp. 243–250.

BlueObelisk (2008). *Properties of the elements.* Atom element properties from elements.xml distributed at http://sourceforge.net/projects/bodr.

Cabral, B., Cam, N. & Foran, J. (1994). *1994 Symposium on Volume Visualization*, pp. 91–98.

Cheng, H.-L. & Shi, X. (2004). *Proceedings of the IEEE Conference on Visualization 2004*, pp. 481–488.

Cipriano, G. & Gleicher, M. (2007). *IEEE Trans. Vis. Comput. Graph.* **13**, 1608–1615.

Ebert, D. S., Musgrave, K. F., Peachey, D., Perlin, K. & Worley, S. (2002). *Texturing and Modeling: a Procedural Approach*, 3rd ed., *The Morgan Kaufmann Series in Computer Graphics*. San Francisco: Morgan Kaufmann.

Ebert, D. S. & Rheingans, P. (2000). *Proceedings of the IEEE Conference on Visualization 2000*, pp. 195–202.

Grottel, S., Reina, G., Vrabec, J. & Ertl, T. (2007). *IEEE Trans. Vis. Comput. Graph.* **13**, 1624–1631.

Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K. & Gross, M. H. (2005). *Comput. Graph. Forum*, **24**, 303–312.

Hu, M., Chen, W., Zhang, T. & Peng, Q. (2006). *Comput. Graph. Int.* pp. 397–403.

Jang, Y., Botchen, R. P., Lauser, A., Ebert, D. S., Gaither, K. P. & Ertl, T. (2006). *Comput. Graph. Forum*, **25**, 587–596.

Jang, Y., Weiler, M., Hopf, M., Huang, J., Ebert, D. S., Gaither, K. P. & Ertl, T. (2004). *EG/IEEE TCVG Symposium on Visualization, VisSym '04*, pp. 35–44, p. 339.

Krüger, J. & Westermann, R. (2003). *Proceedings of the IEEE Conference on Visualization 2003*, pp. 287–292.

Lampe, O. D., Viola, I., Reuter, N. & Hauser, H. (2007). *IEEE Trans. Vis. Comput. Graph.* **13**, 1616–1623.

Lee, C. H. & Varshney, A. (2002). *SPIE Conference on Visualization and Data Analysis*, pp. 80–90.

Liu, F., Liu, Y. & Fordham, D. (2008). *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, pp. 5298–5301.

Loop, C. & Blinn, J. (2006). *ACM Trans. Graph.* **25**, 664–670.

Mehta, S., Hazzard, K., Machiraju, R., Parthasarathy, S. & Wilkins, J. (2004). *Proceedings of the IEEE Conference on Visualization 2004*, pp. 465–472.

Mehta, K. & Jankun-Kelly, T. (2006). *IEEE Trans. Vis. Comput. Graph.* **12**, 1045–1051.

Molekel (2009). *Multiplatform Molecular Visualization*, http://cscs.ch/molekel.

NVIDIA (2009). *Cg – The Language for High-Performance Realtime Graphics*, http://developer.nvidia.com/page/cg_main.html.

Qiao, W., Ebert, D. S., Entezari, A., Korkusinski, M. & Klimeck, G. (2005). *Proceedings of the IEEE Conference on Visualization 2005*, pp. 319–326.

Qiao, W., McLennan, M., Kennell, R., Ebert, D. S. & Klimeck, G. (2006). *IEEE Trans. Vis. Comput. Graph.* **12**, 1061–1068.

Rezk-Salama, C. & Kolb, A. (2005). *Proceedings of Vision, Modeling and Visualization (VMV)*, pp. 115–122.

Röttger, S., Guthe, S., Weiskopf, D., Ertl, T. & Strasser, W. (2003). *VISSYM '03: Proceedings of the Symposium on Data Visualization*, 2003, pp. 231–238.

Schaftenaar, G. & Noordik, J. (2000). *J. Comput. Aided Mol. Des.* **14**, 123–134.

Schmidt-Ehrenberg, J., Baum, D. & Hege, H. C. (2002). *Proceedings of the IEEE Conference on Visualization 2002*, pp. 235–242.

Stone, J. E., Saam, J., Hardy, D. J., Vandivort, K. L., Hwu, Wm. W. & Schulten, K. (2009). *IACAT, GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pp. 9–18. ACM.

Svakhine, N., Jang, Y., Ebert, D. S. & Gaither, K. P. (2005). *Proceedings of the IEEE Conference on Visualization 2005*, pp. 687–694.

Tarini, M., Cignoni, P. & Montani, C. (2006). *IEEE Trans. Vis. Comput. Graph.* **12**, 1237–1244.

Ufimtsev, I. S. & Martinez, T. J. (2008). *J. Chem. Theory Comput.* **4**, 222–231.

Wilson, O., Gelder, A. V. & Wilhelms, J. (1994). *Direct Volume Rendering via Three-Dimensional Textures*. Technical report UCSC-CRL-94–19. University of California – Santa Cruz, USA.