

Interactive Global and Local Deformations for Virtual Clay

Guillaume Dewaele and Marie-Paule Cani
GRAVIR-IMAG*

Abstract

Making virtual modeling as easy and intuitive as real-clay manipulation is still an unsolved problem. This paper takes a step in this direction: in addition to offering standard features such as addition and removal of material, it uses a new real-time plasticity model to let the user apply local and global deformations such as those made in real clay by pressing with a finger or bending a sculpture with one hands. Although not completely physically accurate, the model exhibits several important features of real clay, namely plasticity, mass conservation, and surface tension effects. These features make the model intuitive, since the user obtains the shapes he expects, as demonstrated by our results.

1 Introduction

Compared to traditional tools for numerical shape modeling, real clay remains a very simple and intuitive way to create complex shapes: even children use clay at school. Many artists prefer expressing themselves with real materials instead of using a computer.

If one could get the benefits of real clay in a computer-based modeling system, one could have the best of both worlds: virtual clay would neither dry nor crack; it could be mutated from softer to dryer states as needed; the artist could pause at any time, and return to work as convenient without worrying about material changes in the interim. Furthermore, gravity would no longer be a problem, so shapes that cannot be made easily with real material would become possible. Finally, the advantages of any computer-based modeling tool would apply: the artist would be able to work at any scale and use any size of tool, simplifying the production of fine details as well as global features, and virtual modeling would allow copy/paste, undo, etc., as well as more clay-specific ideas such as temporarily removing a part of the model to ease the editing of hard-to-reach areas.

Although previous virtual clay models offer real-time interaction and a number of interesting features, most were either restricted to carving and adding material, or adapted solely to surface deformation, not allowing topology modification. And the operations they supported often failed to provide an intuitive interface for modeling, because some of the defining features of clay — its constant volume, its surface tension — were not simulated in the systems.

This paper proposes the first volumetric, real-time virtual clay model which can be both sculpted by adding and removing material, and deformed through interaction with rigid tools. Either global or local, the deformations mimic the effects of the tools on real clay, due to plasticity, mass conservation, and surface tension.

1.1 Related work

Although an impressive number of computer graphics works have used the term *sculpting* [2, 3, 10, 11, 12, 14, 16, 18, 19, 20, 27, 28] or even *clay* [7, 17], none has provided a real-time model that mimics the main feature of real clay, namely the ability to undergo plastic, constant volume deformations in addition to matter addition and removal. The previous approaches fall into three main categories: empirical surface models, empirical volume models, and physically-based models.

The first group of interactive sculpting methods rely on surface representations [3, 16, 18]. Such representations ease the application of interactive deformations, but require the use of complex mechanisms for enabling topological changes. Moreover, these geometric models cannot easily provide the same features as the interaction with a volumetric material, such as constant volume deformation.

Volumetric models [1, 9, 10, 11, 12, 19, 20] are much closer to real clay: most of them are defined using a scalar field, which is either stored in a grid or controlled through primitives such as B-spline volumes. The surface of the object is an isosurface of this field, which plays a similar role than a density of material. This representation is very adequate to the application of volumetric operations, such as carving the object or adding material, performed through local editing of field values. These volumetric representations

*GRAVIR is a joint lab of CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier. {Guillaume.Dewaele|Marie-Paule.Cani}@imag.fr

therefore handle any kind of topological change without the need of any specific mechanism. Only local deformations with no volume conservation were defined [10]. A solution using cellular automata has been purposed to compute local deformations of a virtual clay [1, 9]. Interactive free-form modeling with volume conservation and topological changes are possible. However, none of these model provides the ability to deform the piece of clay, as an artist would do while putting the limbs of a model in the right posture.

Although physically-based modeling is a good way to ensure intuitive results when deforming shapes, it has hardly been applied to virtual clay models. Real clay lies somewhere between viscous fluids [24] and plastic solids [23]. All previous physically-based clay models [17, 25, 27] relied on the plastic solid representation. A plastic behavior can easily be obtained by extending an elastic model. Typically, an object’s rest shape is given the ability to locally absorb the deformations that exceed a given threshold [23]. In such models, applying the very large deformations that real clay can undergo, including local matter displacement, would require frequent re-meshing of the model. In particular, enabling topological changes (such as separation of matter into several pieces) appears incompatible with real-time performance, since most real-time deformable models exploit the material’s internal structure for preinverting matrices or pre-computing hierarchies [6, 8, 13, 15]. Therefore, none of the previous physically-based clay models allows the simulation of very large deformations, including relative matter displacement inside the shape, carving, fusion and separation. Geometric topological changes were however made possible using a cleverly adapted representation, i.e. subdivision solids [17]. But these changes in topology still required the user to specify which cells should be deleted or joined.

If we look on the fluid modeling side, some non-real-time particle-based models were used to represent very viscous material [5, 24, 26], somewhat similar to clay. Real-time performance was only obtained in Eulerian fluid simulations [21, 22], for which the fluid was occupying the entire space (in contrast to clay, which is bounded by a time-varying surface).

1.2 Overview

As we just saw, none of the previous real-time deformable model exhibits the features of clay we would like to mimic. The main contribution of this paper is to describe a new deformable model for virtual clay that *does* capture these features. Although it was inspired by the way real material behaves, our model does not fully capture the physics of clay. We believe that its capability to exhibit the main characteristic of clay — extreme plasticity, including en-

abling relative matter displacement inside the block of material, topological changes, constant volume, surface tension — will be sufficient for the user to recover the feeling he gets when interacting with real clay.

Our model exploits a layered approach for achieving this goal with real-time performance: based on a standard volumetric representation (i.e., mass density values stored in a grid), it uses the combination of three independent deformation layers for providing the desired behavior. These three layers respectively model large scale deformations, mass conservation (which yields smaller-scale matter displacement), and surface tension. These layers are activated successively during each time step. Since they share the same underlying volumetric representation, coherence is ensured, a layer being immediately aware of the changes of the clay state due to the action of the other layers.

The remainder of this paper develops as follows: Section 2 describes our representation for clay, and motivates the use of multiple layers for the simulation. Section 3 details our three layers. Section 4 deals with real-time rendering. Section 5 presents results. We conclude and discuss future work in the last section.

2 Our Clay Model

2.1 Volumetric representation

Since we are working with a solid material, a volumetric representation is a natural choice. It permits easy modeling of topological changes, which is quite important if we want to be able to carve and deform any kind of shape.

We thus describe our material with a scalar field function defined on 3-space. It is discretized on a regular grid, and values at non-grid-point locations are interpolated from neighboring grid-point values. In standard implicit modeling [4], field values have no intrinsic meaning: they serve only to distinguish inside from outside, and adjusting them by any positive factor makes no difference to the model. One can use this freedom to store additional information in the scalar field, such as the distance to the level-zero surface [19]. Since we are trying to simulate material behavior, we choose instead to give this field function a different meaning: it represents the density of matter within the object. More precisely, each value on a grid node corresponds to the quantity of matter in a cubic cell surrounding this node. When this value is zero, the cell is void.

Since we want to model incompressible virtual clay, each cell shall only contain a limited quantity of matter. By convention, a field value equal to 1 denotes a full cell. The surface of the clay is defined as an iso-surface of mass density, i.e. we are inside the material when the density reaches a given threshold between 0 and 1.

Adding clay material (respectively removing clay) is done by increasing (respectively decreasing) the field values in regions covered by a tool, exactly as was done in previous approaches [10, 12, 19]. Similarly, smoothing and undo-redo operators can be easily implemented; we currently use the algorithms described in [10].

The remainder of this paper focuses on our new contribution: the ability to deform the clay in real-time, without modifying the underlying quantity of material.

2.2 Deforming virtual clay

In our representation, deforming clay just requires decreasing the field function in some regions, while increasing it in others. Since the quantity of material should not be modified, we basically have to add to some cells the exact quantity of matter we are removing elsewhere, thus modeling matter displacement.

Since no previous physical-based model succeeded in simulating a material similar to clay in real-time, our approach is to directly model the main observed features of real clay, i.e., to take a *phenomenological* approach:

- Real-clay mainly undergoes totally damped, plastic deformations. Although some dynamic effects and limited-range elasticity can be observed in real-clay, we found these negligible in practice.
- Clay is incompressible, and thus preserves its volume during deformation.
- Although some small pieces may separate from the block of material during interaction, the material usually holds together, a behavior we can attribute to a certain amount of surface tension.

Mimicking these features can be done by combining three layers, each modeling a specific aspect of the clay behavior. The first layer handles large-scale effects. This lets the user bend or twist parts of the sculpture. A second layer enforces volume conservation, which may not be exactly ensured by the first layer. Volume preservation will yield local effects resembling footprints — dented in the middle, raised at the edges — in areas where a tool pushes the clay. Finally, the third layer acts as a surface tension mechanism, in order to keep the material as compact as possible. These layers could be activated separately, since they all directly edit the volumetric representation of the clay. But their combination is necessary for achieving the natural clay behavior we are looking for. The next section details the three layers.

3 Simulation of the virtual clay

3.1 Large scale displacements

An natural way to simulate large scale deformations of clay is to rely on the previous work on elastic and plastic models. This could for instance be done by attaching a mass-spring network (or an FEM), at each time step, to the current configuration of the clay, and use its deformation to move material. The underlying volumetric representation would then serve as a basis to ease locating and re-meshing the material. This approach, which we tried first, quickly raised some efficiency issues: while standard elastic and plastic models always use the same network of nodes, which enables optimizations, we had to re-compute the network at each time step, as our sculpture experienced very large deformations and topological changes. Results were thus far from real-time, even using a simple spring model and implicit integration. Moreover, the dynamic behavior of masses and springs was unnecessary, since clay can be seen as a fully damped material.

We therefore looked for a much faster method, one that could provide in one step the same kind of displacement field the mass-spring network would have reached at equilibrium, even if its accuracy would be limited. We found inspiration in fluid mechanics: when an element in a viscous fluid moves, it pulls nearby fluid elements with it. In fact, the quantity of movement diffuses in the fluid: faraway elements are pulled less than nearby ones. We thus decided to use a similar approach, by diffusing inside the clay the displacements dictated by user-controlled tools.

At each time step, when a tool intersects the clay after moving towards it, we calculate the exact time of the collision and the tool displacement since the collision occurred. We also look for the cells covered by the tool. Clay in those cells is moved by the tool, and that movement is further transmitted from cell to cell, moving clay that is farther away from the tool.

We use a propagation scheme inside the clay to get all the cells that will be affected by the tool's motion. And we compute a "pseudo-distance" to the tool for each cell affected. This pseudo-distance is not supposed to approximate the Euclidean distance. It rather represents the local "influence" of the tool: the greater this factor, the less the movement of the tool has impact on the cell.

Starting from the cells close to the tool, we compute the distance from cells to cells until we reach the borders of the object, fixed cells, or cells attached to other tools. The distance for a cell is computed simply by adding $\frac{1}{density}$ to the minimal distance already computed for close cells. Inside of the object, the result obtained is similar to the Manhattan distance (although it doesn't compute distance along straight lines but rather follows the shape of the object). But

in areas where the density is low, the influence of the tool decrease more quickly. This will favor tears near the weak points of the object.

We then compute a displacement field for all the cells under the influence of the tool. When a single tool is used, we can either give to all the matter the exact tool's displacement (which will rigidly move the closest connected part of the clay with the tool), or decrease the amplitude of displacement according to the pseudo-distance to the tool (this way, matter far from the tool will not move). However, in the latter case, the user would have to tune the way the displacement decreases, and restricting deformations to the desired regions may be nonintuitive.

In practice, we do not use this second solution, since, as in real life, the user always uses a support to hold a part of the clay when sculpting. This support can, for instance, represent a table on which the model lies, a wall towards which it is compressed or fingers holding the clay, but may also consist of any set of selected cells inside the object in which matter displacement is clamped to zero. Whatever its shape and number of connected components, the fixed region is treated as a second, immobile tool.

3.1.1 Combining tools and static regions

Suppose that we have cells that are under the influence of two (or even more) tools. The movement of the clay in those cells will be a combination of the movements defined by the two tools. Matter close to the first tool should have the same movement as this tool, and the same for clay in cells close to the other tool. Clay in between should have a intermediate interpolated movement. We use the pseudo-distance to the different tools, introduced in the previous section, to measure their respective influence on a given cell.

Let us call d_1 the pseudo-distance from the cell to the first tool and d_2 the pseudo-distance to the second one. Let d_{12} be the pseudo-distance between the tools (i.e., the lowest pseudo-distance we get between a cell attached to the first tool and one attached to the second one). We use the coefficient $k = \frac{d_1 - d_2}{d_{12}}$ to express the relative influence of the two tools on the cell. For cells where $k = 0$, the two tools have the same influence. When $k < 0$, matter in the cell is more influenced by the first tool's motion than by the second one's motion.

We apply to the matter in each cell a displacement vector δ that is a linear combination of the displacement vectors δ_1 and δ_2 of the two tools as follow:

$$\delta = \frac{1 - k}{2} \delta_1 + \frac{1 + k}{2} \delta_2.$$

In our current implementation, only a single tool is moving, since we use only a single input device. All others tools are static. The user can also select cells where matter can't

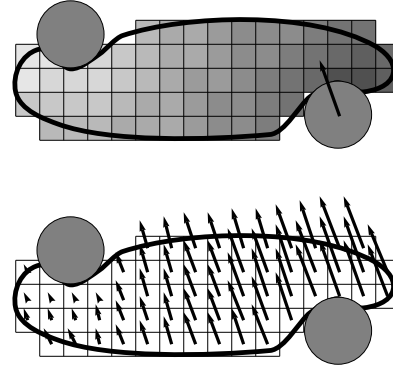


Figure 1. Respective influences of a moving tool and a static tool inside an object and the corresponding displacement field for clay.

move. We compute the influence of all those static tools in the clay, then the influence of the moving tool, and we use the previous equation with $\delta_2 = 0$ (corresponding to the non-motion of the static tools) and δ_1 corresponding to the displacement of the tool currently manipulated by the user (see figure 1).

3.1.2 Moving matter

Once the displacement vector in each cell is evaluated, we only have to move matter according to this displacement field. We remove matter from each cell where the field is non-zero, and put it in the cell whose position is δ relative to the original cell. In general, this new position will not be the center of a grid cell, so the matter is distributed among the eight surrounding nodes, the closest ones receiving the greatest contributions. We simply use a linear interpolation: if (x, y, z) is the destination point and (x_n, y_n, z_n) is the position of one of the eight closest nodes, this node will receive the quantity of clay

$$q_n = q * \left(1 - \frac{\|x - x_n\|}{d_x}\right) * \left(1 - \frac{\|y - y_n\|}{d_y}\right) * \left(1 - \frac{\|z - z_n\|}{d_z}\right)$$

where q is the total quantity of clay we want to move, and d_x , d_y and d_z are the spacings of our grid cells along the three axis. This is illustrated by figure 2.

Unfortunately, the displacement field just computed does not satisfy the condition $div(\delta) = 0$, so we would normally see changes in the volume of the object. The volume conservation layer of our simulation will address this.

3.1.3 Changing the behavior of clay

We have found it useful to have a way to change the behavior of our clay. Real clay can contain a variable quantity

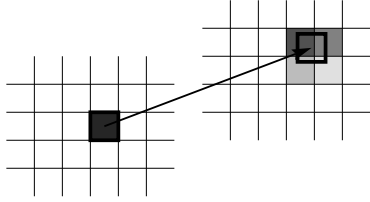


Figure 2. Moving clay between cells.

of water, and thus its behavior can range from plastic to almost fluid. With a very fluid clay, the tool doesn't transmit its movement to the clay as well as it does with plastic clay.

We have two ways of adjusting the clay behavior. The first simply adds a constant to the pseudo-distance between a cell and the tools. This way, the movement of any tool will be less efficiently transmitted to the clay.

The second solution consists of using a fraction α of the tool displacement, where α lies between 0 and 1. Results are quite similar. We currently use this second solution, because even if it's not correct (it lacks, for example, symmetry between static and moving tools), it's easier for the user to understand and to use: a factor near 1 gives a very dry clay; a factor near zero gives a very fluid behavior.

3.2 Mass-conservation layer

3.2.1 Principles

The mass-conservation layer of the simulation aims at enforcing volume conservation. It also models local matter displacements near the surface of the object due to the tool's action. It will result in prints when the user pushes the tool on the object, in folds, etc. Of course, none of these effects can be produced by the previous layer. Indeed, the clay needs to locally move laterally and then even in the opposite direction from the tool to create bumps and folds around it.

The idea behind this layer is quite simple: if, in a cell, the density is greater than the maximum allowed value, 1, the excess is distributed into the six closest cells. When those cells are not full, the process terminates. If they have an excess of matter, they will distribute it among their own closest cells, and so on. Matter will move from cells to cells and finally reach the object's border, where it will find some room to remain. We will see that the object inflates in those areas.

We found the ideas behind this layer in fluid mechanics. When the medium sees locally an excess of pressure (i.e. an excess of matter), we get motions of the fluid from the areas with high pressures to areas with lower ones, until a uniform pressure is obtained. The main difference is that we only consider excess with regard to the maximum density and do

not compare it to the surrounding values. This way, our clay remains solid, and doesn't tend to occupy the whole space.

3.2.2 Interaction with tools

Now we need to see how tools can interact with our mass-conservation layer. We want the tool to push the clay in front of it when the user presses the tool against the object. The interaction is quite straightforward: where we have a tool, there's no more room for matter. The cells covered by the tool cannot contain clay anymore, so all the clay in those cells is in excess. We use the process we just described for moving this matter.

Rather than using purely rigid tools, we limit aliasing artifacts by defining them using a density function. The tool's density decreases near its edges. When the tool occupies eighty percent of a cell (i.e. its density value in the cell is 0.8), there is room for twenty percent of clay. Thus the carved object will have the same roughness as the tool. It is possible, too, to use a previously sculpted piece of clay as a tool. We thus let the user design his own complex tools, for example to be able to make prints or bas reliefs.

A small problem remains. If we simply move matter inside the tool to all close cells, some clay can go through the whole tool and exit on the other side. We thus add one more rule for interacting with tools: clay inside tools can only move outwards. For each cell occupied by the tool, we define allowed and forbidden directions among the six possible directions to nearby cells. This way, tools really push matter in front of them, and no clay goes through the center of the tool.

For efficiency reasons, we precompute those allowed directions when we design a tool. This is done by looking for the closest direction to the surface of the tool. We *could* simply use the (discrete) gradient of the tool's field function. But this will not work for tools sculpted within our system, since we clamped the field value to 1 inside the tool. We need a second field function, with no clamping value this time, so that the gradient can be meaningfully computed anywhere. If we have only a field function already clamped to 1 to describe the tool, we have to build this second field function. This can be done by using a propagation scheme starting from the edges of the tool, and going inside. We use the same algorithm we described in the large-scale displacement algorithm to compute the influence of the tools, except we got rid of the $1/density$ term. This way, we have everywhere an estimation of the distance to the surface of the tool, and the gradient points towards the outer part of the tool. This computation is performed each time we convert a piece of clay into a new tool.

Moving clay in one direction is allowed if this direction makes an angle with the direction of the gradient under a given threshold. We choose to use a 60 degree angle. We

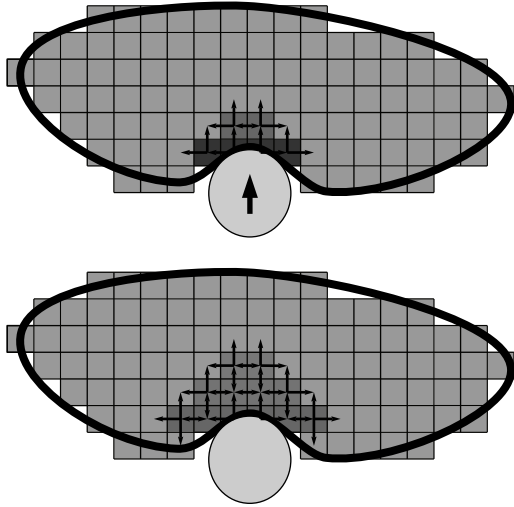


Figure 3. Movements of clay inside the object, following a movement of the tool.

normalize gradient, and we compare its components to 0.5 and -0.5 to decide whether motion along the x -, y -, and z -axes should be allowed.

3.2.3 Implementation issues

To implement efficiently, we simply use a file (i.e. a first-in-first-out queue) which contains all the cells which could have an excess of matter. So we take the first cell of the file, check for an excess, and if there's one, move it in the closest cells and put these cells in the file. We added a flag to each cell to prevent a cell from being in the file more than once.

We do this until the file is empty, or we reach a given number of iterations. This could seem expensive computationally, but it is a very simple scheme, so we can easily do a few hundred of thousands of iterations. One important thing to see is that even if we stop iterating, matter will not be lost: we will simply finish the work later, at the next simulation step.

For convergence, we consider that there's no more excess in a cell as soon as a cell contains less than a threshold a bit higher than 1 (for example we can use 1.01). The little variations in the volume of the object would not be noticed by the user. This threshold can be adjusted if we want more precision or faster simulation.

Another thing has been done to get a faster result: we don't want clay to go from a cell to another, and then come back to the first cell. We could try to remember from where the clay was coming, but it simply don't work because clay can arrive in a cell from several directions. Instead, we move more clay than needed, taking into account that one sixth of it will come back in the cell.

3.3 Surface tension

After several deformations using the two layers above, the matter tends to become less and less compact. Clay pushed by the tools can indeed be dispersed around the object, and the transition from inside (density equal to 1) to outside (void cells) gets slower and slower. One of the problems with cells with low densities is that the user does not see them, so strange effects can arise if a tool pushes these small quantities of clay in front of it: clay popping from nowhere when density, due to action of the tool, rises to the threshold; inaccurate changes of the surface location, etc. Moreover, since matter in cells of low density is no longer visible, the object's volume will seem to decrease, even if matter does not really disappear.

The surface-tension layer tries to resolve and avoid these problems. It keeps the gradient of density near the surface of the clay to an acceptable value. Matter in cells with very low densities is moved to nearby cells with higher densities. We look for every cell with a value below a threshold¹. At each such cell, we compute the gradient of the field function by using finite differences with nearby cells. If the length of the gradient is below another threshold (which correspond to the gradient we would like to have near the surface of the object), we move clay from the cell with a low density to closest cells with higher ones. This way, the object remains compact. The layer can be seen as adding a surface tension effect for a fluid.

While the previous layer prevents contraction of our clay, this layer tries to avoid expansion. This will separate the object in two different compact parts when the user tries to stretch it too far. Indeed, if you try to cut an object in two, the area between the two pieces will have a decrease of density. When the middle area density falls below our threshold, the matter is divided between the two parts, and the object is eventually cut in half.

Even with surface tension, some very small pieces of matter may separate from the main block of clay, like crumbs from a piece of toast; these are sets of a few neighboring cells with above-threshold densities. This is still physically correct, and the user should not be surprised, since these crumbs are visible. But because they can be distracting for the user, we get rid of them as soon as possible by removing them from the working space. If we want to preserve the volume of the object, we can put the matter removed this way back in the closest cell with high density, as if the crumb had been eaten up by the clay.

¹In our implementation, we use 0.3, and an isovalue of 0.6

4 Real-time rendering

4.1 Off-line rendering

Most of the time, we simply use the marching cubes algorithm to render the clay surface. Because connectivity can change during deformations or carving, we need to explore the whole space to get all the disconnected parts of the object. As this is computationally expensive, we can't do it for each frame. So we only update triangles in regions where the density has changed, as it was already suggested [12]. Each time a cell's density is modified, the eight cubes around the node are marked as changed. When the scene must be displayed, all the cubes that have been marked are updated, and triangles inside of them are changed. Unfortunately, when the user bends a large part of the object, this method becomes quite expensive computation-wise, since a lot of cells will be updated. In order to save as much computing time as possible for the simulation loop, we designed a fast rendering method to take care of such situations.

4.2 Fast rendering during interaction

Our fast rendering technique is based on raycasting. We throw rays along one of the axes of the scene (e.g. the z -axis) and we search for the first intersection of the ray with the surface. Since we throw rays along an axis of the discretization grid, we only need to look for the first node of each row parallel to the axis which contains a value above the threshold. The exact point of intersection with the surface can be obtained by linear interpolation between this node and the previous one.

We build this way a heightfield $z = f(x, y)$. The result we obtain looks like a cloth thrown on the objects. Of course, we do not have the back side of the object anymore, nor the parts behind others. So we have to choose an axis close to the camera direction.

Directly rendering the heightfield would not be a good solution: When we have two objects, one of them partially behind the other, they would be merged (see figure 4). So we instead refine our surface by casting additional rays in areas where z varies quickly, and we only draw triangles between points with nearby z -values. This way, we get more precise contours, too.

As soon as we have time (when the user removes the tool from the clay, for instance), we update all the cells marked, and go back to marching cube rendering. As we can measure the time taken by computation for the simulation, and we know how many cells have to be updated, it's quite easy to decide when we must switch to fast rendering, and go back to marching cubes.

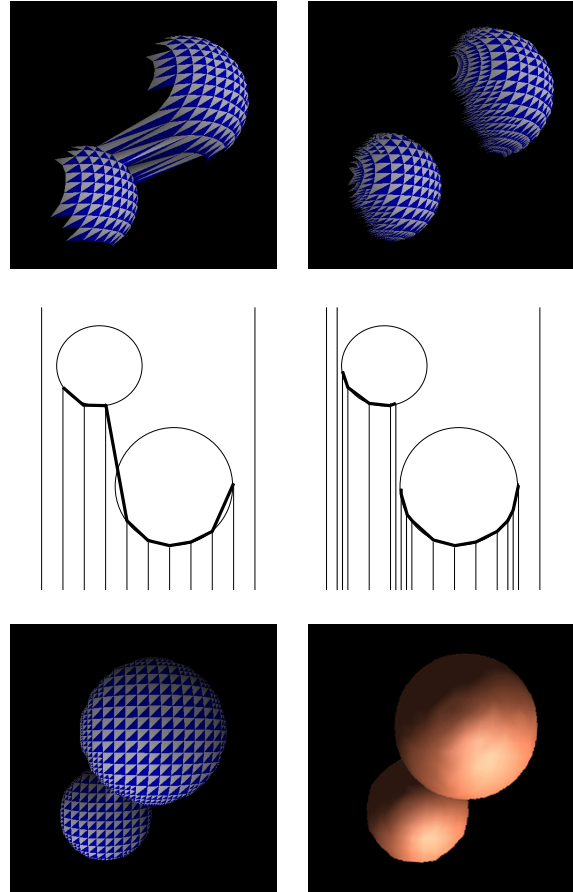


Figure 4. Glitches in upper left picture are solved by casting additional rays. The result obtained by our method is shown below.

4.3 Efficient use of information on the surface

The ray casting technique would be quite expensive if we had to do this work at each time step. In practice, we can often keep information about the location where the closest cell can be found. We will only seek this closest cell between two z values, z_{near} and z_{far} , which are indices of cells in the row.

When we find the index z of the closest cell by casting a ray, we take this z for z_{near} and z_{far} . This way, if there's no change in cells along the row between z_{near} and the camera, we won't have to compute that intersection at next time step. And we have some events that change the values of z_{near} and z_{far} .

First, if the density value of a cell on the same row, closer to the camera than z_{near} , has its value rise above the iso-value, we update the value of z_{near} and z_{far} so that they match the z of this changed cell. In fact, we just found the

new closest cell along the row. If the value in a cell located between z_{near} and z_{far} reaches the threshold, we only update z_{far} .

Sometimes, the value of the cell located at z_{far} falls below the threshold. If this arrives, there may be no more cells between z_{near} and z_{far} whose value is above the threshold. So we set z_{far} as the last index of the row to be able to find the intersection (if there is one). It can happen when matter is pushed away from the camera or shifts laterally.

All those changes to z_{near} and z_{far} are done during simulation. When we want to render the image, we simply have to look for the correct z -value of intersection between z_{near} and z_{far} . Then we reset z_{near} and z_{far} for the next time step.

The reason why this approach is quite fast is that we often have to look for z only in very small areas. Where objects don't move or in areas where rays don't intersect the clay, there's no computation to do ; when matter moves towards the camera, we also have no computation, and when it moves away, we'll only check one or two cells. In fact, only lateral shifts can be expensive around the silhouettes, but this will generally only affect a small number of rays.

5 Results

Our results were obtained by letting the user specify regions where the clay is static, place some static tools, and then move matter using a single rigid tool attached to the mouse. The three layers were simulated once, in the order we presented them, for each computation step.

Rather than showing the design of a complex object, which would hide the real intuitiveness of intermediate deformations, our set of results instead aim at showing that we obtain the effect we are looking for in a set of simple cases, for which we know how some real clay would behave. Results can be seen on figure 5, and videos sequences are available on the web (<http://www.imagis.imag.fr/Publications/2003/DC03>).

5.1 Bending

The first example illustrates plastic deformations, where the user tries to bend a part of the sculpture. We choose to use a bar of clay for this example. We simply put a first tool in the contact of the clay, which will be used as a support to bend the bar. Then we move a second tool towards the object, which pushes the clay and thus bends the bar. The result is close to what we could expect of real clay.

5.2 Holes

Our system allows us to dig holes in the clay. We take a thin layer of clay, and we fix the four corners of the layer

(we could instead have used four tools behind the object as supports). Then we push our tool towards the clay. At first the tool creates a small fold and begins to deform the piece of clay. But soon the tool goes through the clay, creating a hole (the last images of the hole sequences are taken from the opposite side).

We can adjust the difficulty of digging in the object by adjusting the fluidity of the clay. For example, if the user wants to dig a hole in a thick object, as seen in the next example, he should use a more fluid clay. This way, digging a hole without deforming the object too much becomes possible. One can see that a bump behind the block is created when the tool exits the object. This is a normal consequence of volume conservation, and similar to what could be produced with real clay.

5.3 Folds and prints

This is a very common behavior of our virtual clay: as soon as we move a tool towards the clay, the clay, pushed by the tool, creates folds. This can be seen in our next example, where we use a sphere of clay fixed in its center. For the last test, we used a tool with a more complex shape, to show that our model can make prints of any shape, simply by pressing a tool against the clay. Once again, to make prints more easily, the user use a fluid behavior for the clay.

5.4 Performances

Computing a framerate for our system is quite difficult, since it's heavily related to the user's actions. The previous examples were sculpted on an Athlon 1.5 GHz, 512MHz or RAM, on a 32x32x32 grid. We managed to get about fifty simulation steps per second, rendering half of them.

We also used an older computer, a Pentium III 500MHz which provided interactive response, even though the framerate was limited by the lack of graphic acceleration to about 10-15 frames per second. Memory seems to have quite a large influence on performance.

6 Conclusion

We have described a real-time virtual clay model which can both be sculpted, as in previous work, by adding and removing material, and deformed through the interaction with rigid tools. Among the real-clay features, the ability to apply these deformations is essential, since it enables the artist to create complex shapes in a simple configuration, and then deform them to a more intricate posture, while preserving the volume and the relative lengths of different regions.

Our deformations, both large and small scale, mimic the effects of tools on real clay (i.e., a plastic material with a constant mass and some surface tension). Therefore it is

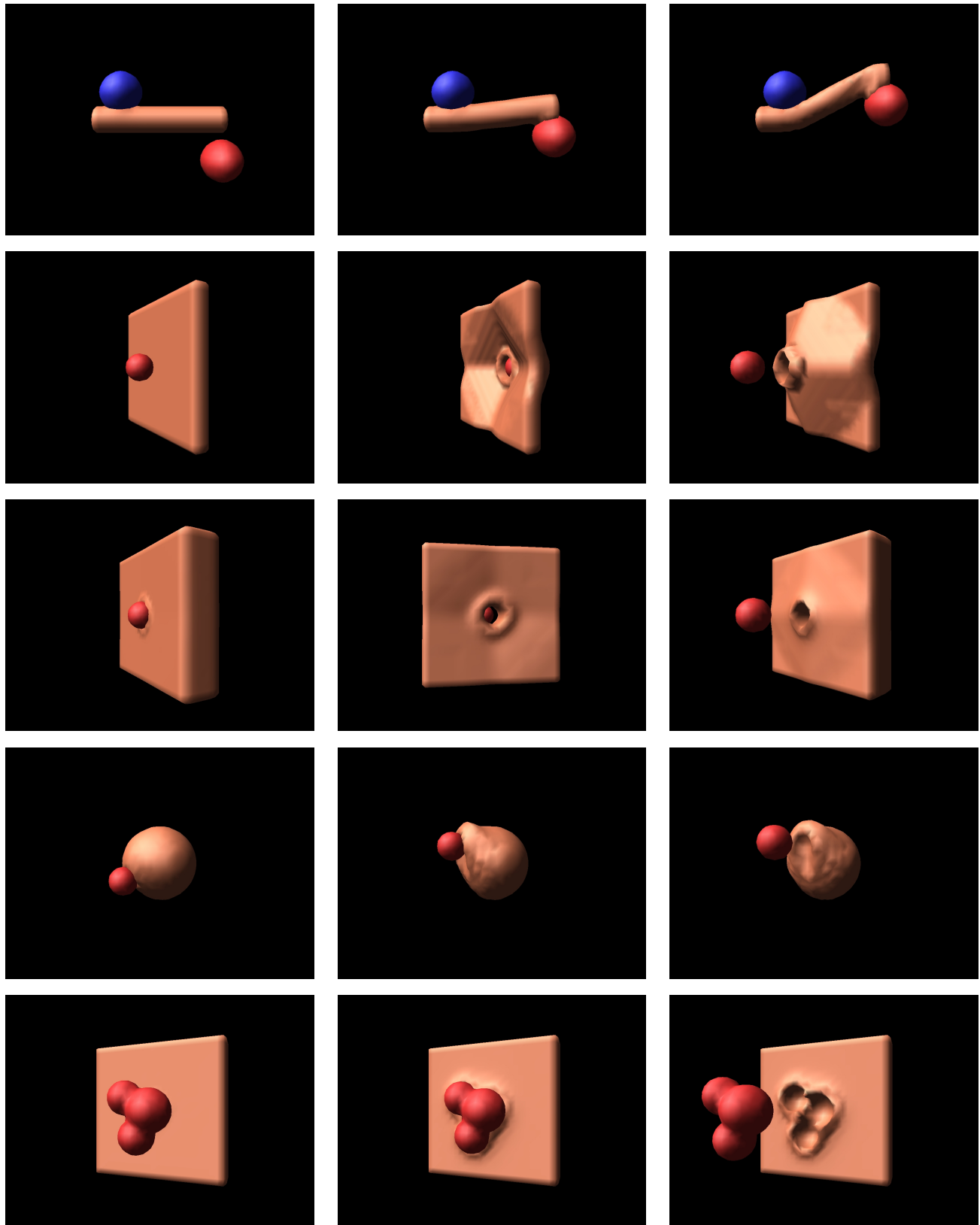


Figure 5. Examples of results obtained with our system. See section 5 for details

easy to achieve such usual behaviors as local folds, footprints, and large scale bending when a thin part, held at one extremity, is pushed.

Although inspired by physical properties of real-material, our model is not truly “physically-based”. We believe that the ability to exhibit the main characteristics of real-clay at very low computational cost (and hence in real-time) is sufficient to make the user recover the feeling he gets by interacting with real dough or clay.

Our future work will focus on new ways of specifying user interaction, and providing some force feedback, another essential element for the user to re-cover the gestures he uses in real life.

Acknowledgements : We would like to thank John Hughes for re-reading this paper.

References

- [1] H. Arata, Y. Takai, N. K. Takai, and T. Yamamoto. Free-form shape modeling by 3d cellular automata. In *International Conference on Shape Modeling and Applications*, pages 242–247, 1999.
- [2] A. Bærentzen. Octree-based volume sculpting. Presented at *IEEE Visualization '98*, 1998. <http://www.gk.dtu.dk/Andreas/publications.html>.
- [3] J.-R. Bill and S. Lodha. Sculpting polygonal models using virtual tools. In *Graphics Interface '95*, pages 272–278, Quebec, Canada, May 1995.
- [4] J. Bloomenthal, C. Bajaj, J. Blinn, M.-P. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufman, 1997.
- [5] M.-P. Cani and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):39–50, Mar. 1997.
- [6] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. A multiresolution framework for dynamic deformations. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 41–48, July 2002.
- [7] Y.-H. Chai, G. Luecke, and J. Edwards. Virtual clay modeling using the isu exoskeleton. *Proceedings of VRAIS'98*, Mar. 1998.
- [8] G. DeBunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 31–36, August 2001.
- [9] S. Druon, A. Crosnier, and L. Brigandat. Efficient cellular automata for 2d / 3d free-form modeling. *WSCG*, 11, Feb. 2003.
- [10] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *the Visual Computer*, 16(8):469–480, Dec. 2000. A preliminary version of this paper appeared in *Implicit Surfaces'99*, Bordeaux, France, sept 1999.
- [11] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Resolution adaptive volume sculpting. *Graphical Models (GMOD)*, 63:459–478, march 2002. Special Issue on Volume Modelling.
- [12] T. Galyean and J. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics*, 25(4):267–274, July 1991. Proceedings of SIGGRAPH'91 (Las Vegas, Nevada, July 1991).
- [13] E. Grinspun, P. Krysl, and P. Schröder. Charms: A simple framework for adaptive simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002. Proceedings of ACM SIGGRAPH 2002.
- [14] F. D. IX, J. El-Sana, H. Qin, and A. Kaufman. Haptic sculpting of dynamic surfaces. *1999 ACM Symposium on Interactive 3D Graphics*, pages 103–110, April 1999. ISBN 1-58113-082-1.
- [15] D. L. James and D. K. Pai. Artdefo - accurate real time deformable objects. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 65–72, Los Angeles, California, August 1999. ACM SIGGRAPH / Addison Wesley Longman.
- [16] E. K.-Y. Jeng and Z. Xiang. Moving cursor plane for interactive sculpting. *ACM Transactions on Graphics*, 15(3):211–222, July 1996.
- [17] K. T. McDonnell, H. Qin, and R. A. Wlodarczyk. Virtual clay: A real-time sculpting system with haptic toolkits. *2001 ACM Symposium on Interactive 3D Graphics*, pages 179–190, March 2001. ISBN 1-58113-292-1.
- [18] B. Naylor. Sculpt: An interactive solid modeling tool. *Proceedings of Graphics Interface '90*, pages 138–148, 1990.
- [19] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. *Proceedings of SIGGRAPH 2001*, pages 47–56, August 2001. ISBN 1-58113-292-1.
- [20] A. Raviv and G. Elber. Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design*, 32(8-9):513–526, August 2000. ISSN 0010-4485, an early version of this paper appeared in the Proceedings of the fifth symposium on Solid modeling and applications, 1999, pages 246–257.
- [21] J. Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 121–128, Aug. 1999.
- [22] J. Stam. A simple fluid solver based on the fft. *Journal of Graphics Tools*, 6(2):43–52, 2001.
- [23] D. Terzopoulos and K. Fleischer. Modeling inelastic deformations: Viscoelasticity, plasticity, fracture. *Computer Graphics*, 22(4):269–278, Aug. 1988. Proceedings of SIGGRAPH'88 (Atlanta).
- [24] D. Terzopoulos, J. Platt, and K. Fleischer. Heating and melting deformable models (from goop to glop). *Proceedings of Graphic Interface '89*, pages 219–226, 1989.
- [25] D. Terzopoulos and H. Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, Apr. 1994.
- [26] D. Tonnesen. Modeling liquids and solids using thermal particles. In *Graphics Interface '91*, pages 255–262, Calgary, AL, June 1991.
- [27] T. Vassilev. Interactive sculpting with deformable nonuniform b-splines. *Computer Graphics Forum*, 16(4):191–199, 1997.
- [28] S. W. Wang and A. E. Kaufman. Volume sculpting. *1995 Symposium on Interactive 3D Graphics*, pages 151–156, April 1995. ISBN 0-89791-736-7.