An Interactive Visualisation Tool Applied to the Simulation of Glass Pressing

K.Laevsky, A. Telea, R.M.M. Mattheij

Department of Mathematics and Computer Science,

Eindhoven University of Technology,

PO Box 5613, 5600 MB The Netherlands

Abstract

An interactive numerical simulation approach to the process of glass pressing is presented. Glass is modelled as a strongly viscous Newtonian fluid whose deformation under pressure is described by a Stokes equation. Modelling the evolution of the glass free boundary in time poses a particular problem which is dealt with by a special integration technique. Next, we present how we integrate the numerical simulation with an interactive visualisation and computational steering tool. The resulting application allows for interactive simulation control and result monitoring. The presented software integration technique does not require any changes to the numerical simulation code and can be thus used to couple other similar computational engines with a steering and visualisation front-end.

1 Introduction

An important tool in scientific computing is the visualisation of numerical simulations. Traditionally the simulation is performed by a numerical package and the visualisation is seen as a separate process of producing one or more pictures from the computed data sets. For practical use this is often cumbersome, as insight in a time-dependent process implies the ability of the user to interactively control the simulation parameters and monitor its results. A second problem is that program modification, such as

inserting a new numerical solver or mesh generator, or changing the visualisation method, usually requires manual editing and recompilation of the source code. In contrast to the above, one would rather like to have a simple, interactive way to monitor, control and construct the simulation and visualisation program, e.g. by means of a graphics-user interface driven tool.

This paper describes such a system for a specific application: the pressing of glass in a mould. In this problem one wishes to predict the flow development (in terms of velocity and pressure quantities) of a gob of glass under pressure in a confinement. The ultimate goal is to obtain an optimal mould shape (geometry) based on the pressing simulation. The simulation evolution is affected by many parameters such as initial velocity, mould geometry, etc. An interactive simulation and visualisation tool is thus essential for solving the inverse problem of assessing the optimal mould shape, by providing steering and result monitoring facilities.

In order to understand the pressing process, the mathematical model used is first described in Section 2. Next we discuss how we solve the reslting equations numerically (Section 3). The rest of the paper is devoted to the description of an interactive visualisation tool for the glass simulation. In Section 4 the advantages of steering simulation and visualisation systems as opposed to non-interactive systems are outlined. Section 5 shows how the glass pressing problem has been implemented in the interactive simulation and visualisation system VISSION. Section 6 illustrates the potential of our approach by showing various results. We conclude the paper with a discussion of some further possibilities.

2 Modelling the Process

This section describes the morphology process for producing packing glass, such as bottles or jars. From the oven a gob of glass is being transported to a mould and there it is pressed into a preform by a plunger which is moving upward. The result of this is a so called *parison*, which is an intermediate product only. The next and final step is the blowing phase where this parison is brought to its final shape by air pressure (see Fig.2.1). We shall only consider the pressing phase here.

The pressing of glass in a mould is still a complicated process. In order to simplify our discussion below we will neglect the influence of the temperature on the flow. Since the viscosity of the material is strongly dependent on the temperature, one may wonder whether this is realistic. One can show,

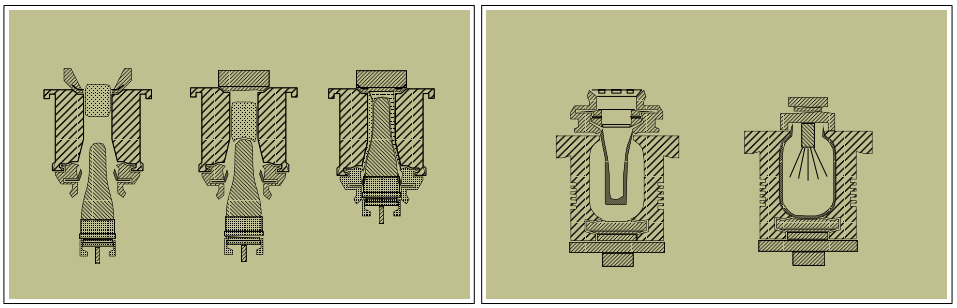


Figure 2.1: Production process.

however, that the actual flow of the glass is nearly isothermal in most cases, because of the low conductivity. As a consequence the heat exchange effectively takes places only after the motion has stopped. Actually, in the present problem that kind of a decoupling is a bit more complicated. One can assume that heat may arise because of friction if the viscous forces are high. But it is not the case here (see [1]). Hence, in order to describe the morphology of the glass we consider only the motion equations with corresponding boundary conditions, which define the velocity field and the pressure. Moreover we will assume the glass to be an incompressible Newtonian fluid.

Typical values for the problem under consideration are:

$$\begin{aligned} \eta &= 10^4 \ \text{kg/sec m} && - && \text{the dynamic viscosity of the glass,} \\ \rho &= 2.5 \cdot 10^3 \ \text{kg/m}^3 && - && \text{the density of glass,} \\ T &= 10^{-1} \ \text{sec} && - && \text{the typical pressing ,} \\ L &= 10^{-2} \ \text{m} && - && \text{the typical scale for the parison,} \end{aligned} \tag{2.1}$$

and finally the typical velocity which is dependent on T and L:

$$U := L/T = 10^{-1} \ \text{m/sec.}$$

Let Ω_t be the region occupied by the liquid at time $t \in [0, T]$. Denote the velocity and the stress tensor by $\mathbf{v}$ and σ respectively. Consider then the Navier-Stokes equations for incompressible fluids in

the time dependent domain Ω_t:

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) - \nabla\cdot\sigma &= \rho\mathbf{f}, \qquad \text{in } \Omega_t, \\ \nabla\cdot\mathbf{v} &= 0, \qquad \text{in } \Omega_t, \end{aligned} \tag{2.2}$$

where ρ is the mass density and $\mathbf{f}$ are the volume forces. The stress tensor σ is related to velocity gradient $\nabla\mathbf{v}$, pressure p and dynamic viscosity μ as follows:

$$\sigma = -p\mathrm{I} + \mu\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) \tag{2.3}$$

Substitution of (2.3) into (2.2) gives us the following equations

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) &= \rho\mathbf{f} + \nabla p - \mu\nabla^2\mathbf{v} \quad \text{in } \Omega_t, \\ \nabla\cdot\mathbf{v} &= 0, \qquad \text{in } \Omega_t. \end{aligned} \tag{2.4}$$

The same equations can be rewritten in dimensionless form. Using the definitions in (2.1) we have:

$$\mathbf{v} = \frac{L}{T}\mathbf{v}', \quad p = \frac{1}{Re}\rho U^2 p', \quad Re = \frac{\rho U L}{\eta},$$

$$\mathbf{x} = L\mathbf{x}', \quad t = \frac{L}{U}t', \quad \mu = \eta\mu',$$

where Re is the *Reynolds number* and $\mathbf{v}'$, p', $\mathbf{x}'$, t' are dimensionless variables. Then the previous system of equations (2.4) reads as follows:

$$\begin{aligned} Re\left(\frac{\partial \mathbf{v}'}{\partial t'} + \mathbf{v}'\cdot\nabla\mathbf{v}'\right) &= Re\frac{L}{U^2}\mathbf{f} + \nabla p' - \mu'\nabla^2\mathbf{v}' \quad \text{in } \Omega_t, \\ \nabla\cdot\mathbf{v}' &= 0, \qquad \text{in } \Omega_t. \end{aligned} \tag{2.5}$$

The volume forces consist of the force of gravity only, i.e.

$$\|\mathbf{f}\| \approx 10 \text{ kg m/sec}^2$$

According to (2.1) the Reynolds number for the problem is approximately 10^{-4}; hence the left-hand side of 2.5 is sufficiently small and can thus effectively be skipped from the equation. According to

the previous remark the term which includes the volume forces is approximately 10^{-3}, which is also negligible. This all means that the viscous forces dominate the volume forces.

As a result we obtain the Stokes equations for an incompressible fluid. Ommiting the prime they read

$$\begin{aligned} \mu\nabla^2\mathbf{v} - \nabla p &= 0 \qquad \text{in } \Omega_t, \\ \nabla\cdot\mathbf{v} &= 0, \qquad \text{in } \Omega_t. \end{aligned} \tag{2.6}$$

This system of equations together with boundary conditions define the velocity field and the pressure.

Although the initial form of the gob may not necessarily be axisymmetric, it is reasonable to employ the axisymmetric geometry of mould and plunger and assume the flow as such to be axisymmetric as well; see Fig. 2.2 a,b. Hence it can be reduced to a two dimensional case. The domain Ω_t will be associated now with the configuration in Fig. 2.2 b.

Let $\Gamma_t = \partial\Omega_t$ be the boundary of Ω_t. It is easy to see that Γ_t consists of four parts:

$$\Gamma_t = \Gamma_m \cup \Gamma_f \cup \Gamma_p \cup \Gamma_s,$$

corresponding to the mould, free boundary, plunger, and symmetric part respectively.

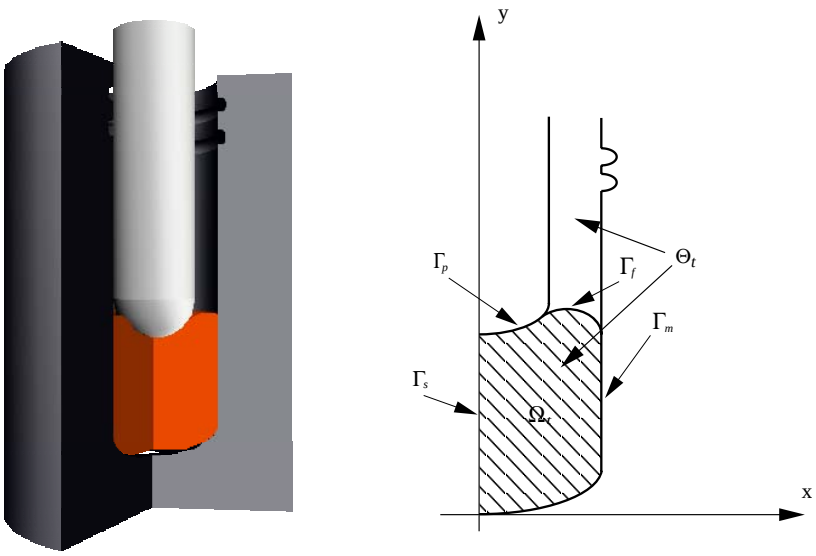
Assuming slip boundary conditions on the corresponding parts Γ_m, Γ_p of the boundary Γ_t we have

$$\begin{aligned} \mathbf{v}\cdot\mathbf{n} &= 0, \\ \alpha\,\mathbf{v}\cdot\mathbf{t} &= (1-\alpha)(\sigma\mathbf{n})\cdot\mathbf{t}, \end{aligned} \tag{2.7}$$

on Γ_m, and

$$\begin{aligned} \mathbf{v}\cdot\mathbf{n} &= \mathbf{v}_p\cdot\mathbf{n}, \\ \beta\,\mathbf{v}\cdot\mathbf{t} &= \beta\,\mathbf{v}_p\cdot\mathbf{t} + (1-\beta)(\sigma\mathbf{n})\cdot\mathbf{t}, \end{aligned} \tag{2.8}$$

on Γ_p, where $\mathbf{v}_p := (0, V)^T$ is the velocity of the plunger. Both in (2.7), (2.8) the slip parameters α, β are from the interval $[0, 1]$. It is easy to see that maximum value of the parameter corresponds to the no-slip boundary conditions, i.e. for $\alpha = 1$ we have $\mathbf{v} = 0$ on Γ_m, and for $\beta = 1$ we have $\mathbf{v} = \mathbf{v}_p$ on Γ_p.



a) b)

Figure 2.2: Problem domain.

At the free boundary, the normal stress must be equal to external pressure p_0, which is assumed to be constant. The tangential stress must be equal to zero. Hence:

$$\begin{aligned} (\sigma \mathbf{n}) \cdot \mathbf{n} &= p_0, \\ (\sigma \mathbf{n}) \cdot \mathbf{t} &= 0. \end{aligned} \tag{2.9}$$

For the symmetry boundary Γ_s the normal component of velocity must be equal to zero, as well as the tangential component of stress vector $\sigma \mathbf{n}$:

$$\begin{aligned} \mathbf{v} \cdot \mathbf{n} &= 0, \\ (\sigma \mathbf{n}) \cdot \mathbf{t} &= 0. \end{aligned}$$

One should note that (2.6) is not a stationary problem since we have the kinematic boundary condition (2.8). Indeed, the domain Ω_t, corresponding to the region occupied with glass at time t, is time-dependent and changes during the process. In the next section we will discuss how we deal with this problem numerically.

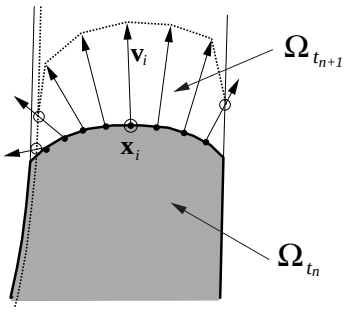


Figure 3.1: Clip algorithm.

3 Numerical approach

The Stokes equations (2.6) together with the boundary conditions (2.7-2.10) can be solved by a finite element code. One should note that the geometry of Ω_t is defined by mould and plunger, except for the free part Γ_f. The approximation of Γ_f requires a special technique which we shall describe now. First we note that the aforementioned Stokes problem, although stationary, has a kinematic boundary condition 2.8. The resulting velocity field $\mathbf{v}$ for this geometry can be used to predict the next position of the free boundary. More precisely, let $\mathbf{x} : [0, T] \times \Omega_0 \to \mathbb{R}^2$ be such a mapping that:

$$\mathbf{x}(0) = \Omega_0, \quad \mathbf{x}(t) = \Omega_t, \tag{3.1}$$

where Ω_t is the problem domain as defined before. Then the relation between velocity field and the domain geometry can be described by the initial value problem:

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= \mathbf{v}(\mathbf{x}(t)), \quad t \in [0, T], \\ \mathbf{x}(0) &= \Omega_0. \end{aligned} \tag{3.2}$$

The velocity field $\mathbf{v}(\mathbf{x}(t))$ can be obtained by solving the Stokes equations in Ω_t. However, one should realise that the geometry of Ω_t depends on that velocity field.

In order to overcome this problem we will use the following strategy. Let us define

$$t_n = n\Delta t, \quad n = 1, \ldots, N,$$

such that $t_0 = 0$, $t_N = T$. After discretisation and solving Stokes equations with corresponding boundary conditions in Ω_{t_n} (which are assumed to be defined) we obtain the velocity field $\mathbf{v}^n$. Instead

of (3.2) we solve the initial value problem (3.3):

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= \mathbf{v}^n, \quad t \in [t_n, t_{n+1}], \\ \mathbf{x}(t_n) &= \Omega_{t_n}. \end{aligned} \tag{3.3}$$

In particular for any point $\mathbf{x}_i^n$ at the free boundary Γ_f we may consider this as a Lagrangian displacement, which we may e.g. discretise by the explicit Euler method:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^n. \tag{3.4}$$

The global error for this algorithm is of the first order.

The geometry of $\Omega_{t_{n+1}}$ can be obtained now, and hence the boundary conditions required for solving the flow equations at t_{n+1} can be defined. The same procedure is repeated then until the final geometry Ω_{t_N} and corresponding flow quantities have been computed. One should note that Ω_{t_n}, $n = 1, \ldots, N$ according to (3.3) is an approximation of original mapping (3.1). Instead of Euler explicit it is possible to use more sophisticated integration schemes. For our present problem setting however, we will be satisfied with (3.4).

Consider now in more detail the deformation of the free boundary during a time step. Applying formula (3.4) for a point $\mathbf{x}_i^n$ at the boundary Γ_f^n (i.e. the boundary Γ_f at time t_n) with corresponding velocities $\mathbf{v}_i^n$, we see that some of the points $\mathbf{x}_i^{n+1}$ don't belong to the domain as defined by the mould and the plunger. Let us denote the latter by $\Theta_{t_{n+1}}$ (see Fig. 2.2 b). This configuration is changed explicitly by moving the plunger at each time iteration. We now simply clip displacements outside this $\Theta_{t_{n+1}}$, see Fig. 3.1. So the position of $\mathbf{x}_i^{n+1}$ is defined now by intersection of $\mathbf{x}_i^n + \Delta t \mathbf{v}_i^n$ and $\Theta_{t_{n+1}}$:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \alpha_i \Delta t \mathbf{v}_i^n, \quad \alpha_i \in (0, 1]. \tag{3.5}$$

Where α_i is chosen such that

$$\Omega_{t_{n+1}} \subset \Theta_{t_{n+1}}.$$

We shall call this algorithm the "clip" algorithm. For the velocities that must be clipped ($\alpha_i < 1$) the error is apparently $O(h^2)$! The actual values of α_i depend on the characteristics of the process, Δt and

the mesh size h. In a practical implementation the term $(1-\alpha_i)\|\dot{\mathbf{x}}_i(t_n)\|$ should be first order in Δt, as is also the actual order of the explicit Euler method. One can analyse this in a more detailed way. However, we shall stick to this approach that allows the fast computations needed by our interactive tool presented next.

4 Interactive Visualisation Tool Design

We now turn to our implementational platform. To begin with, we note that there are two important dimensions of computer based simulations, namely controlling and interacting with the simulation on one hand, and examining (e.g. by direct visualisation) the generated data on the other. Combined in a flexible way, the above features lead to software tools which help conveying a better insight and understanding of the evolution of the simulated process.

Based on the way they address the above process control and data interrogation requirements, simulation systems range from non-interactive systems to steering systems (see [11, 15]). Most such systems implement the same simulation scenario consisting of a problem definition phase, followed by the numerical computations phase, and finally the result interrogation phase In our case, these phases correspond with the initial mould geometry and velocity setup, solving the Stokes problem and moving the boundary at every simulation time step (Section 3), and visualising the obtained mould geometry.

Non-interactive systems implement the above three phases in a loosely coupled, unsynchronized manner, usually as one or several batch applications run separately by the user who manually feeds the output of an application to the input of the next in terms of files (Fig. 4.1 a). Interactivity is limited to configuration file editing, and simulating time dependent processes is reduced to running the above pipeline manually to produce a set of output files corresponding to input data at different time instants. Such systems can thus convey only a limited insight in time-dependent processes, as the time-dependent behaviour is manually simulated by the user. The advantage of the non-interactive system model is its loose coupling, meaning that various independently developed software applications can cooperate without having to reprogram them, based on an input-output file compatibility. As most numerical simulation software comes as batch monolithic applications whose interactivity is limited to reading and writing files, the non-interactive system simulation model is still the most widespread.

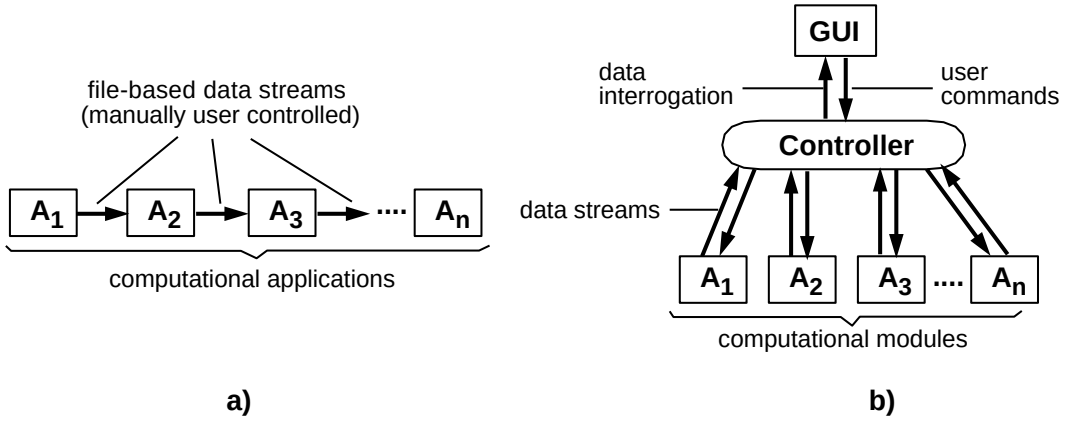


Figure 4.1: Non-interactive (a) and steering (b) simulation systems

Steering systems are the other implementation extreme [10, 4, 8, 13, 12, 14]. They treat the three simulation phases uniformly, providing interactive ways to control all the problem definition, computation, and result visualisation phases (usually by means of graphics user interfaces (GUIs)), and a pipeline synchronisation controller that makes a stage compute automatically as soon as the previous stage has produced its result (Fig. 4.1 b). Direct control over all the parameters enables the user to investigate the process parameter space easily, tune the computational controls on the fly, and monitor on-line the process evolution in time. However, implementing a steering model for simulations raises two main problems:

1. most steering systems have a single control thread, meaning that their interactivity is bounded by the speed of the slowest stage. The numerical computations are however rarely real-time, like the Stokes problem solving involved in our glass pressing simulation. The interactivity of all the stages (including e.g. tuning the parameters of the faster visualisation stage) is thus brought practically to zero. This diminishes considerably the attractivity of using steering systems to integrate the computational stage.

2. to provide uniform control, synchronisation, and GUI policy for all stages, steering systems require these stages to conform to various software interfaces. This often implies fundamental modifications of the numerical simulation code requiring extensive programming knowledge. This makes the integration option less attractive for non-programming experts such as numeri-

cal analysts.

5 An Interactive Visualisation Application for the Glass Pressing Simulation

We have developed a software application that integrates an interactive visualisation back-end with a numerical simulation of the glass pressing process. The numerical simulation was designed and implemented as a stand-alone monolithic aplication, to which an interactive visualisation environment was further coupled. The resulting application allows for an automatic monitoring of the time-dependent numerical data produced by the finite element simulation and offers an interactive way to choose and tune various data visualisation methods.

The first part of the pipeline (Fig. 5.1) is a classical example of monolithic finite-element application written in Fortran that performs the computational domain discretisation followed by the iterative solving of the time-dependent Stokes equations amd boundary displacement for a sequence of time moments. The second part of the pipeline consists of the general-purpose object-oriented environment for scientific visualisation VISSION [7]. VISSION is based on the dataflow model ([5, 4]), in which visualisation or data processing tasks are described as networks of computational modules communicating by reading, processing, and writing data to each other. VISSION provides an interactive way to construct a visualisation or data processing task by visually assembling module icons picked from an available module icon library to create the desired network. Next, various data sets can be fed into the network to be processed (e.g. compute gradients of scalar fields, extract isosurfaces or trace streamlines in datasets), visualized in various ways (e.g. surfaces, flow ribbons, elevation plots, stream tubes, wireframe meshes, etc) and manipulated in 2D or 3D interactive viewers. The user can also interactively change the parameters of the network modules via several GUIs to e.g. select a new isosurface threshold, zoom/pan the data viewers, or interactively probe the datasets to find their values in some points of interest.

We could easily couple the numerical simulation stage with the visualisation environment back-end without having to modify a single line of the numerical software by using the so-called *data sensors*

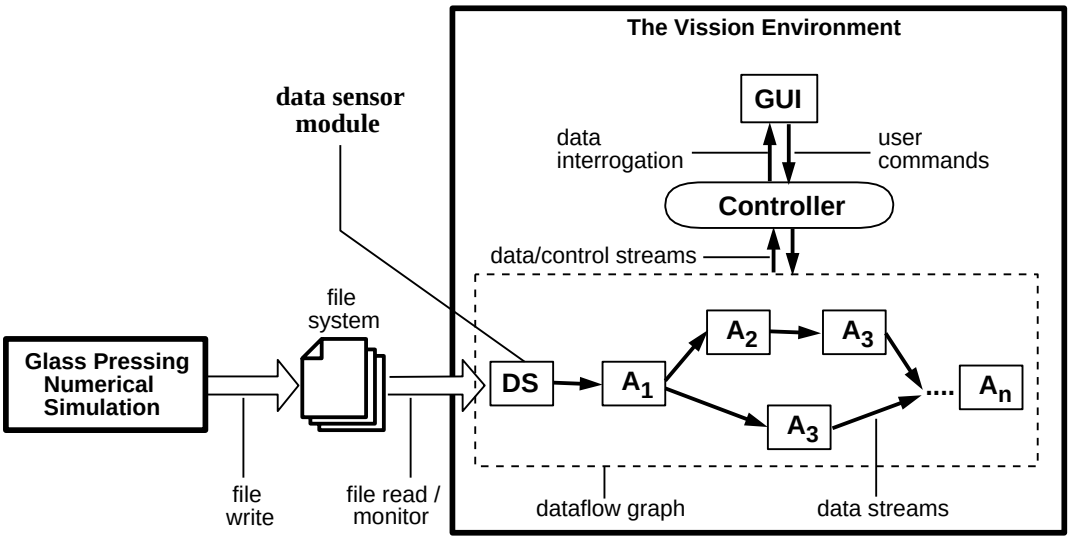


Figure 5.1: Pipeline for interactive visualisation of the glass pressing simulation

of VISSION. These are special modules which, placed at the beginning of dataflow networks, monitor some desired data files for specific changes such as file creation or modification. As the numerical application completes a new simulation time step, it writes the new computational mesh and solution data (glass pressure and velocity in our case) as a new file or set of files, or modifies existing ones. When this event happens, VISSION's data sensors fire and the new data is read in the input of the visualisation network (Fig. 6.1) which automatically executes presenting a new view to the user. The whole process executes as though the data flows transparently from the numerical solvers up to the visualisation viewer modules, producing an effect similar with the one obtained if the numerical simulation had been integrated in the dataflow network as a VISSION module.

Coupling the numerical simulation with VISSION via data sensors has several advantages:

1. the numerical simulation and the VISSION environment can run either on the same machine or on different workstations sharing the same file system transparently (see Fig. 5.2 a). This was very useful as we could run the whole application on a single machine or use a powerful machine for performing the numerical computations and a separate fast graphics workstation for performing the visualisation.

2. the second advantage of the data sensor coupling resides in the fact that the numerical simulation

and the visualisation have independent control threads. The user can interactively choose several visualisation modules (i.e. edit VISSION's dataflow network), tune their parameters on-line, and see the changes in the visualized images almost instantly, as the visualisation modules are very fast, while the numerical simulation keeps computing at its own, usually slower pace. When a new time step is ready, the new data 'flows into' VISSION transparently. Similarly with editing or changing the parameters of the visualisation independently of the numerical simulation's pace, one can stop, reconfigure, and restart the numerical application without having to care of the possibly several VISSION sessions monitoring its output.

Combining the monolithic numerical simulation with the interactive visualisation by allowing an independent control thread is an attractive variant of the steering solution, as the inherently interactive part (the visualisation) remains interactive and still synchronized with the slower computational part. This setup was especially convenient in the two-machine scenario described above.

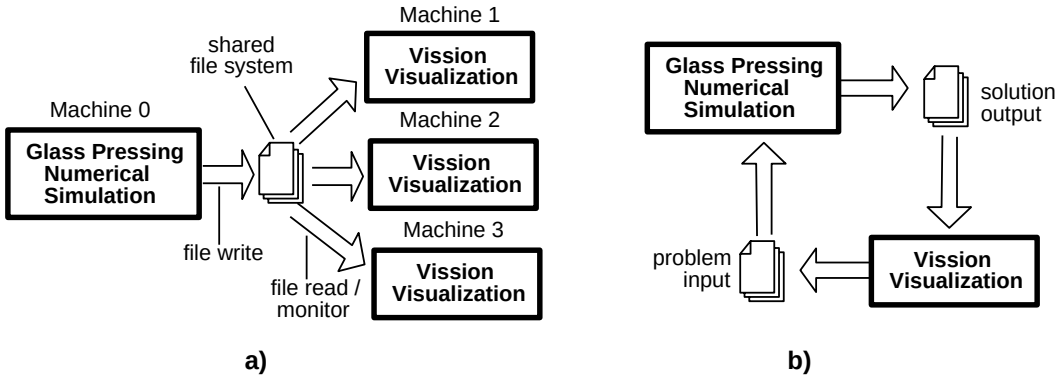


Figure 5.2: Distributed computation and visualisation (a). Steering pipeline combining computation and visualisation (b).

To couple VISSION with the numerical glass simulation (or any similar computational application exporting its data via files or other UNIX mechanisms such as pipes, message queues, or shared memory) we only have to write a simple data reader VISSION module accepting the data file format provided by the numerical code. The object-oriented design of VISSION and its rich support of data set types (unstructured, regular, rectilinear, and curvilinear grids with multidimensional data per point or cell ([5]) allowed us to write such a reader in about one hour. The alternative of integrating the numer-

ical application in VISSION as a computational module would have meant complex reprogramming of the numerical code. The file sensor approach practically extends the dataflow network outside VISSION's boundaries by treating the numerical simulation as an 'external module' at the network's input. Synchronized communication with the numerical application is thus provided transparently, without having to modify its code, by using the file system's file access mechanisms.

Overall, the presented architecture combines the advantages of monolithic applications (reuse without reprogramming) and steering systems (interactivity, several control threads, visual programming, GUIs) presented in Section 4 and allows transparent execution of the numerical computations and visualisation on the same or different machines. In the same time, the mentioned disadvantages of tight integration based on reprogramming are removed by using the data sensor mechanism. The presented mechanism was also used to integrate other simulations with the interactive visualisation environment VISSION.

6 Results

To illustrate the interactive visualisation system described in the previous section, several results are presented. Figure 6.1 presents an overview picture of an interactive glass pressing visualisation in the VISSION environment. In this example, a dataflow network was visually constructed to monitor the output of the glass pressing numerical simulation and display the computed flow velocity magnitude in two data viewers, overlaid with a set of stream lines for the flow field (left data viewer), respectively contour lines of the flow field magnitude (right data viewer). The dataflow network starts with a data sensor module which provides the interactive monitoring of the running numerical simulation (as described in the previous section), followed by a data file reader. When the data sensor is removed from the network, one can use the system as a classical off-line visualisation tool for already computed flow data sets. The reader module loads two data sets computed by the numerical simulator, i.e. a velocity and a pressure field defined on the 2D glass mould cross-section mesh. The rest of the network contains various modules that compute the velocity field magnitude, stream lines, and contour lines from the simulation input data, and present the resulting information to the end-user in the data viewer windows that provide zoom, pan, and rotate facilities. Figure 6.2 shows several images produced with

the visualisation setup described above, displaying the glass velocity magnitude (left column), respectively its pressure (right column) at six different time instants during a pressing simulation. The above application is an exact implementation of the conceptual pipeline presented in Fig. 5.1.

The results of the glass pressing simulation are ultimately to be used in an industrial production environment. In order to convey a more intuitive understanding of the glass pressing process for the industry end-users, a tri-dimensional reconstruction of the process was performed in the VISSION environment. The reconstruction pipeline takes as input the boundaries of the 2D computational meshes produced by the numerical simulation and rotates each of them around their symmetry axis to produce a 3D shape. Next, the obtained shapes are rendered with appropriate lighting and glass-like material properties in the same data viewer we used for the visualisation of the 2D computational data. The end-users can now manipulate the obtained 3D glass shapes directly in the viewers to get a better impression of the simulation results. In order to visualize the results independently on the VISSION environment, a MPEG movie production module was appended to the reconstruction pipeline. This allowed us to automatically create MPEG movies of the glass pressing simulation and of its 3D reconstruction, which can be played back by standard MPEG players independently on the VISSION system. Figure 6.3 shows 12 frames from a movie produced in VISSION using the above method. All the above operations (3D reconstruction, lighting, rendering, and MPEG production) could be done using less than 10 of the standard VISSION modules.

7 Discussion

Several other solutions to the problem of interactive visualisation and steering of scientific computations exist. The Computational Steering Environment (CSE) [10] consists of a synchronisation kernel (the Data Manager) to which several satellites running on the same or different machines can be attached to perform visualisation and computational tasks. The satellites can run asynchronously, the data access being synchronized by the Data Manager. However, the CSE offers no capability to interactively build a visualisation network by visual assembly of modules as VISSION does, nor it supports other dataset representation besides structured grids, making it unflexible for our purposes. Finally, VISSION can easily provide most of the satellite-Data Manager communication facilities by implement-

ing several types of data sensors that monitor various resources (e.g. files, shared memory, message queues, sockets) for various events. The CSE application steering capabilities could be also provided in the context of VISSION's file sensors. For example, we could steer the numerical simulation from VISSION's interactive GUI by making the latter export the parameters it modifies as shared resources (e.g. files or pipes) and making the former read the data in these resources (see Fig. 5.2 b for a sketch of the computational steering loop).

The AVS and Oorange systems ([4, 6]) are also based on the dataflow network and visual network editor concepts similarly to VISSION. However, their multi-programming language design exposes their users to several technicalities of several programming languages, making it harder to program new modules (e.g. data readers or data sensors) for non-programming experts. These isues are detailed by us elsewhere [16].

The Visualisation Toolkit (vtk) system ([5]) offers practically the same data representation classes and ease to write new modules as VISSION (which actually incorporates the vtk modules). However, vtk offers no visual dataflow network editor, nor module GUIs, limiting the run-time user interaction to scripting languages. Moreover, vtk uses a demand-driven network update policy, while AVS, VISSION, and Oorange use an event-driven one (see [5, 7] for details). This would make the implementation of data sensors considerably more difficult in vtk than e.g. in VISSION, where a file sensor code has approximately 50 lines of C++.

References

[1] T.D. CHANDRA, S.W. RIENSTRA, *Analytical Approximation to the Viscous Glass Flow Problem in the Mould-Plunger Pressing Process*, RANA 97-08, Technical University of Eindhoven,1997.

[2] B. STROUSTRUP, *The C++ Programming Manual*, Addison-Wesley,1993.

[3] J. WERNECKE, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*, Addison-Wesley, 1993.

[4] C. UPSON, T. FAULHABER, D. KAMINS, D. LAIDLAW, D. SCHLEGEL, J. VROOM, R. GURWITZ, AND A. VAN DAM, *The Application Visualization System: A Computational Environment for Scientific Visualization.*, IEEE Computer Graphics and Applications, July 1989, 30–42.

[5] W. SCHROEDER, K. MARTIN, B. LORENSEN, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, 1995

[6] C. GUNN, A. ORTMANN, U. PINKALL, K. POLTHIER, U. SCHWARZ, *Oorange: A Virtual Laboratory for Experimental Mathematics*, Sonderforschungsbereich 288, Technical University Berlin. URL http://www-sfb288.math.tu-berlin.de/oorange/OorangeDoc.html

[7] A.C. TELEA, J.J. VAN WIJK, VISSION*: An Object-Oriented Dataflow System for Simulation and Visualization*, in *Proceedings of the VisSym'99 IEEE-Eurographics Symposium on visualization and scientific simulation*, Vienna, Austria

[8] A.C. TELEA, C.W.A.M. VAN OVERVELD, *An Object-Oriented Interactive System for Scientific Simulations: Design and Applications*, in *Mathematical Visualization*, H.-C. Hege and K. Polthier (eds.), Springer Verlag 1998

[9] B. MEYER, *Object-oriented software construction*, Prentice Hall, 1997

[10] J. J. VAN WIJK AND R. VAN LIERE, *An environment for computational steering*, in G. M. Nielson, H. Mueller and H. Hagen, eds, *Scientific Visualization: Overviews, Methodologies and Techniques*, Computer Society Press, 1997

[11] J.E. ROSENBLUM AND R.A. EARNSHAW, EDITORS, *Scientific visualization: advances and challenges*, Academic Press, London, 1997

[12] S. RATHMAYER AND M. LENKE, *A tool for on-line visualization and interactive steering of parallel hpc applications*, in *Proceedings of the 11th International Parallel Processing Symposium*, IPPS 97, 1997

[13] D. JABLONOWSKI, J. D. BRUNER, B. BLISS, AND R. B. HABER, *VASE: The visualization and application steering environment*, in *Proceedings of Supercomputing '93*, pages 560-569, 1993

[14] G. A. GEIST, J. A. KOHL, P. M. PAPADOPOULOS, *CUMULVS: Providing fault tolerance, visualization, and steering of parallel applications*, in *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3): 224-235, 1997

[15] M. J. NOOT, A. C. TELEA, J. K. M. JANSEN, R. M. M. MATTHEIJ, *Real time numerical simulation and visualization of electrochemical drilling*, *Computing and Visualization in Science*, vol. 1, Springer, 1998, pp. 105-111

[16] A. C. TELEA *Combining Object Orientation and Dataflow Modelling in the* VISSION *Simulation System*, in *Proceedings of the TOOLS'99 Europe Conference*, ACM Press, 1999, pp 56-65

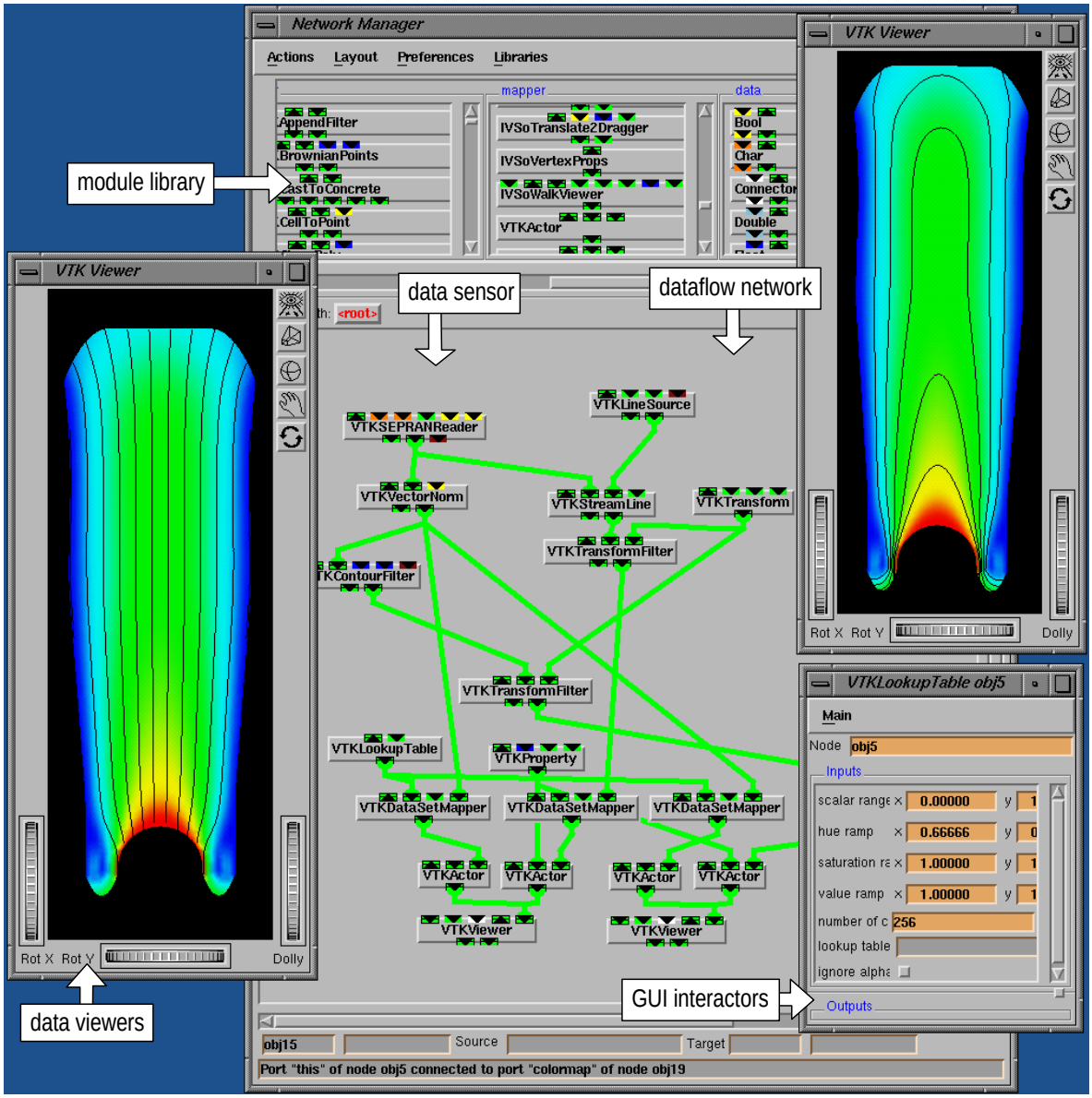


Figure 6.1: Interactive visualisation session for the glass pressing simulation

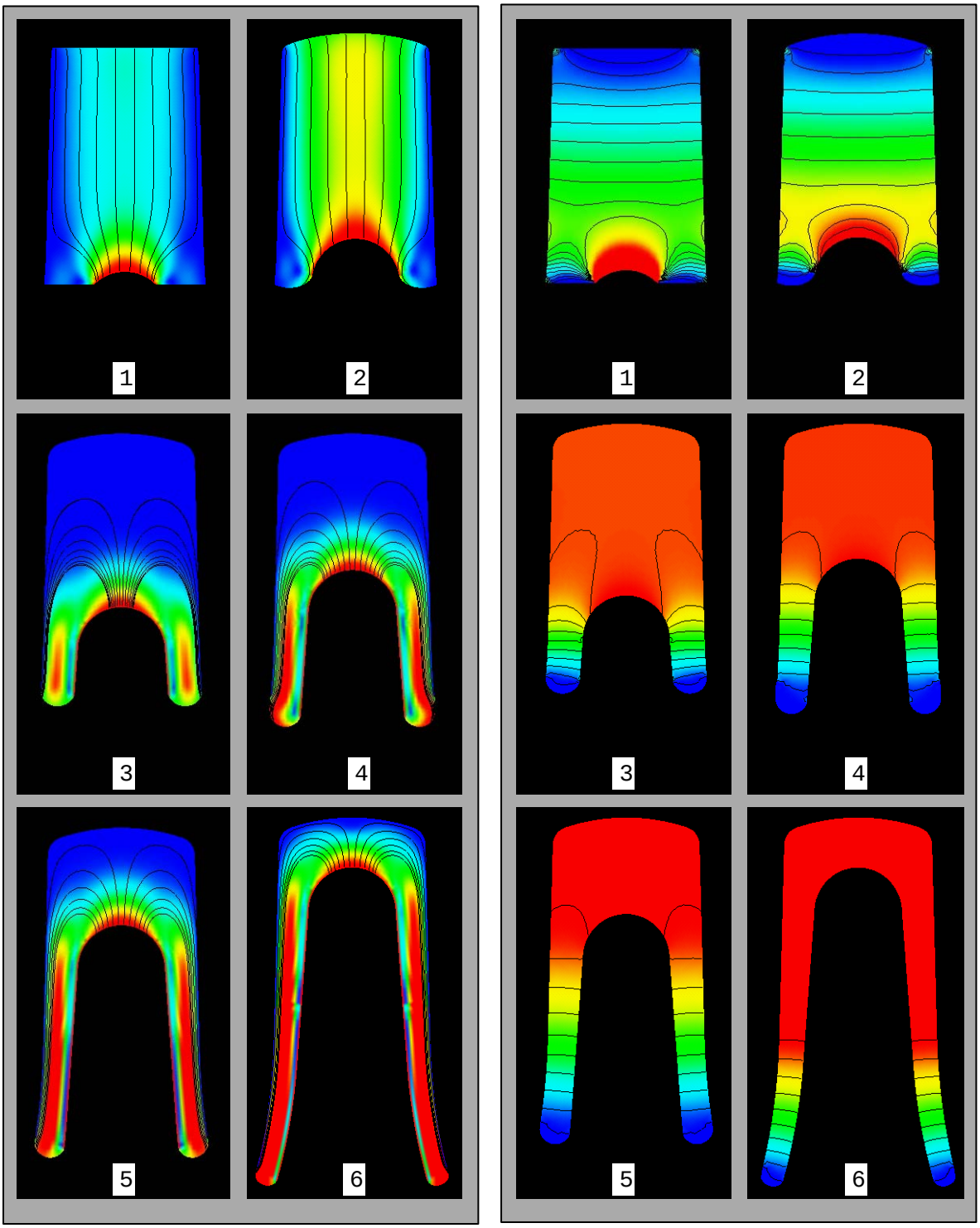


Figure 6.2: Frames from a 3D animation of the glass pressing simulation

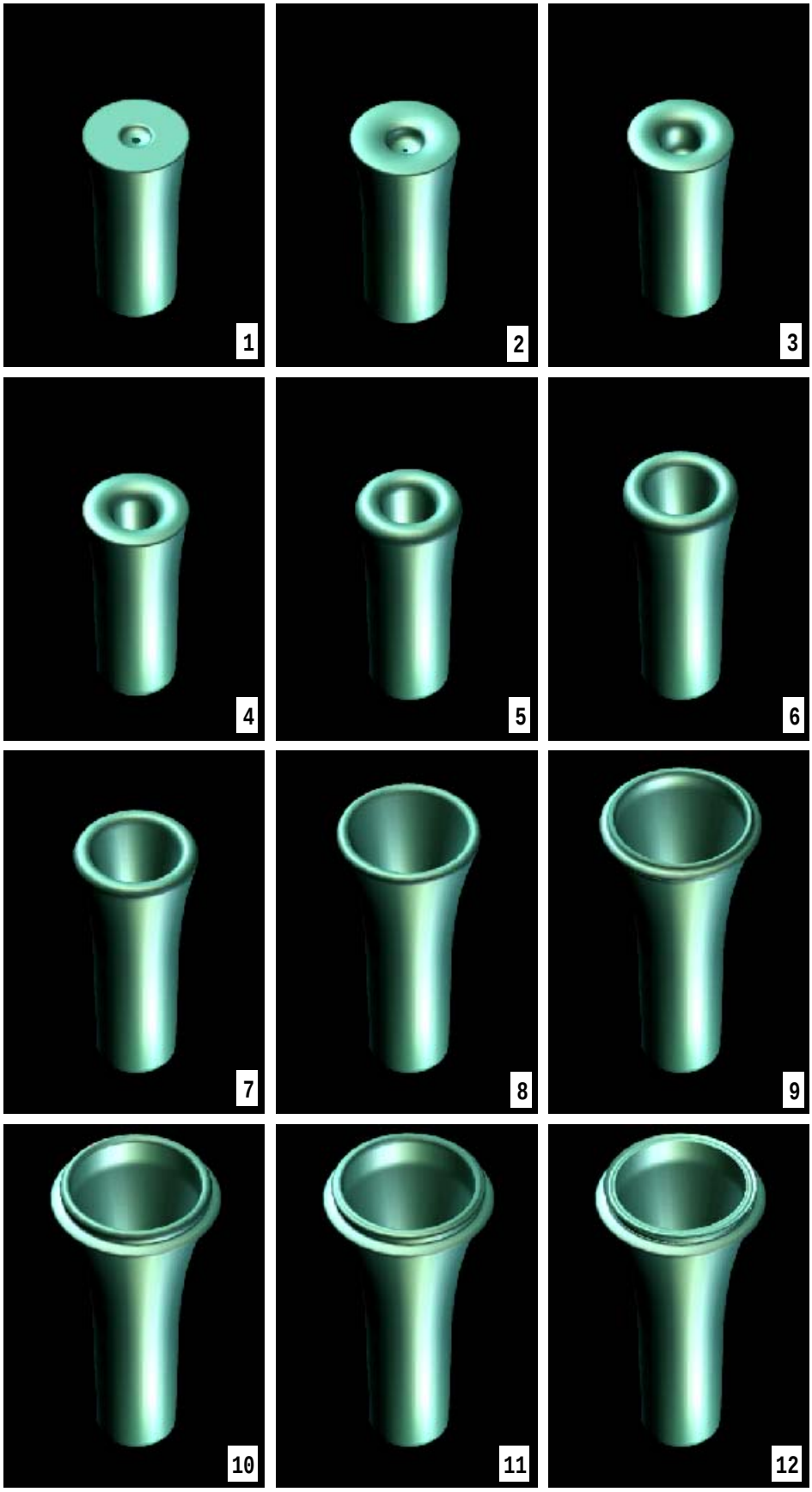


Figure 6.3: Frames from a 3D animation of the glass pressing simulation