

# Real-Time Soft Body Simulation Using Pressure Forces

---

*Patrick Martin*

*7/22/2008*

## Table of Contents

1	Overview .....	4
2	Soft Body Objects .....	4
2.1	Spring Forces .....	4
2.2	Pressure Force .....	4
2.3	Ideal Gas Approximation .....	4
3	Rigid Body Objects .....	5
4	Collision Detection and Resolution .....	5
4.1	Rigid vs. Rigid .....	5
4.1.1	Narrow Phase .....	5
4.1.2	Resolution .....	5
4.2	Rigid vs. Soft .....	6
4.2.1	Narrow Phase .....	6
4.2.2	Resolution .....	6
4.3	Soft vs. Soft .....	6
4.3.1	Narrow Phase .....	6
4.3.2	Resolution .....	6
4.4	Impulse Based Physics .....	7
4.5	Separating Axis Theorem (SAT) .....	7
4.6	Point in Concave Object .....	7
4.7	Point in Convex Object .....	8
4.8	Point in Triangle Test .....	8
5	Updates .....	8
5.1	Soft Body Updates .....	8
5.2	Rigid Body Updates .....	8
6	Known Issues .....	9
7	Future Research .....	9
8	Works Cited .....	10
9	Screen Captures .....	10
10	Appendix A: Equations .....	11
10.1	Cross Product .....	11

10.2	Determinate .....	11
10.3	Dot Product .....	11
10.4	Euler-Cromer .....	11
10.5	Hooke's Law .....	11
10.6	Ideal Gas Law .....	11
10.7	Impulse .....	12
10.8	Inertia Tensor of a Rectangular Prism .....	12
10.9	Inverse of Diagonal Matrix .....	13
10.10	Point in Tetrahedron .....	13
10.11	Pressure Force .....	13
10.12	Runge-Kutta 4 .....	13
10.13	Torque from Linear Force .....	14

## 1 Overview

For this project I attempted to create a soft body simulation that could run in real time and be integrated in games related products. The soft body objects have the ability to collide with each other and various rigid body object in a 3D environment as well as maintain their shape using a combination of spring and pressure forces.

## 2 Soft Body Objects

Soft body objects are treated as a collection of particles, and their surface membrane is formed via a spring system across these particles. The object's 3D shape is maintained via the pressure force pushing outwards from this membrane. This system is very similar to the same system used by (Matyka & Ollila, 2003).

### 2.1 Spring Forces

The surface of a soft body object is formed via a collection of points. Any three adjacent points are then grouped to form a triangle, where each edge of the triangle represents a spring. All spring forces are computed using Hooke's Law, where the force of a spring is defined as  $F_s = -k(x - x_0)$ . In this equation,  $k$  is the spring's strength,  $x$  is the current length of the spring, and  $x_0$  is the resting length of the spring. The spring force acts as a restorative force against the pressure force to prevent an object from indefinitely expanding.

### 2.2 Pressure Force

The pressure force is implemented in a similar fashion to the paper (Matyka & Ollila, 2003). I calculate a pressure force on a per triangle basis as  $\vec{F}_p = P\hat{n}A$ . In this case  $P$  is calculated using the ideal gas approximation,  $\hat{n}$  is the normal of the triangle, and  $A$  is the triangle's area. Since forces in my simulation do not actually act directly on triangles, I distribute this force amongst the triangle's vertices. Therefore, the force that each vertex receives is  $\frac{1}{3}\vec{F}_p$ .

### 2.3 Ideal Gas Approximation

The pressure,  $P$ , is calculated using the ideal gas approximation  $P = V^{-1}nRT$ .  $V$  is the total volume of the object,  $n$  is the number of moles contained within the object,  $R$  is the ideal gas constant (8.314), and  $T$  is the absolute temperature. For the purposes of this simulation,  $nRT$  is treated as a user defined scalar since those variables will not change between the frames without direct user intervention.

The total volume ( $V$ ) is computed using a technique in (Zhang & Chen, 2001). I loop through all the triangles in an object, and calculate the volume of the tetrahedron formed between that triangle and the origin. If any individual triangle in the object is defined via the counter clockwise ordered points  $a$ ,  $b$ , and  $c$  then the volume of the tetrahedron they form is  $V = \frac{1}{6}|-c_x b_y a_z + b_x c_y a_z + c_x a_y b_z - a_x c_y b_z - b_x a_y c_z + a_x b_y c_z|$ . Then this volume is either added to or removed from the total velocity according to the direction of the triangle's normal, with a normal facing away from the origin indicating

addition and a normal facing towards the origin indicating subtraction. Since this triangle is counter clockwise ordered, one can form the two edges  $e_1 = b - a$  and  $e_2 = c - a$ . The normal is therefore calculated via the cross product  $\hat{n} = e_1 \times e_2$ , and the sign is the same sign as the dot product  $a \cdot \hat{n}$ . This dot product works because the position vector to point  $a$  is guaranteed to be pointing away from the origin approximately along the normal and a positive dot product indicates that the normal pointing in the same general direction (with a product of 0 indicating a  $90^\circ$  angle between the normal and  $a$ ).

### 3 Rigid Body Objects

My simulation also implements a traditional (although fairly basic) rigid body simulation. Each object's behavior is determined via an inertia tensor ( $I$ ), mass, linear acceleration ( $a$ ), linear velocity ( $v$ ), position ( $p$ ), torque ( $\tau$ ), angular velocity ( $\omega$ ), and angle ( $\vartheta$ ). Since all my rigid objects are boxes, the

inertia tensor is  $I = \begin{pmatrix} \frac{m}{12}(w^2 + d^2) & 0 & 0 \\ 0 & \frac{m}{12}(h^2 + d^2) & 0 \\ 0 & 0 & \frac{m}{12}(h^2 + w^2) \end{pmatrix}$  which is stored along with the mass

and the inverse of both the inertia and mass.

### 4 Collision Detection and Resolution

Collision pairs are first identified using a broad phase consisting of bounding spheres detection. These resulting collisions are reduced into three classes based on the objects involved: Rigid vs. Rigid, Rigid vs. Soft, and Soft vs. Soft. Once a pair is classified it then goes through one to many unique narrow phase tests and a unique collision resolution scheme.

#### 4.1 Rigid vs. Rigid

##### 4.1.1 Narrow Phase

Since all rigid bodies in my simulation are convex (cubes), narrow phase uses the Separating Axis Theorem (SAT) mentioned below. The SAT will return the point of intersection and the depth of intersection for use in the rigid body collision resolution

##### 4.1.2 Resolution

Collisions between rigid body objects are resolved using an impulse based physics method. Since my simulation uses a very simple method of rigid body simulation, it is only able to track one point of collision between any two objects. If two objects are stacked vertically they would bounce apart at one of the corners, so I use a collision point that is halfway between the two objects. This works fairly well since all objects are cubes of the same size, but is not entirely physically accurate.

## 4.2 Rigid vs. Soft

### 4.2.1 Narrow Phase

Rigid vs. Soft collisions operate on each point that is determined to be intersecting with a face due to the possibility of multiple points of collision from the soft body object. For all the points of the rigid object I perform a point in concave object test and for all the points in the soft object I perform a point in convex object test to determine the point/face pairs.

### 4.2.2 Resolution

Each point is resolved individually with an arbitrary ordering. First, the objects are adjusted so they are no longer interpenetrating. Then velocities at the point of intersection for the soft body and rigid body are calculated, and then a simple bounce calculation is performed with the velocities and masses. Finally, the total change in velocity is calculated for the rigid object, and applied to the object's angular velocity.

The change in velocity is determined along the normal of collision ( $\hat{n}$ ), which is taken from the face in the point/face pair. The rigid object is positioned at point  $p$  and the collision point is point  $q$ . A vector,  $\vec{r}$ , is formed from  $p$  to  $q$  as  $\vec{r} = q - p$ . The linear velocity of the rigid object is labeled as  $\vec{v}$ , and the angular velocity is  $\vec{\omega}$ . This leads to the total velocity of the rigid object at point  $q$  as  $\vec{v}_t = \vec{v} + \vec{\omega} \times \vec{r}$ . The total change in velocity is first converted to angular momentum as  $\vec{L} = \vec{r} \times \vec{v}_t$ . The inertia tensor ( $I$ ) is removed leaving the change in angular velocity as  $\vec{\omega}' = I^{-1}\vec{L}$ . Angular and linear velocity are updated by these new computed values.

## 4.3 Soft vs. Soft

### 4.3.1 Narrow Phase

Since narrow phase detection is extremely expensive, I first perform an SAT test with the objects. Then, I loop through the points first determining if a point in one object lies within the bounding sphere of the other object allowing an early exit if the point does not lie within the intersection of the two objects. Once it is determined that the two objects may be colliding and a point on one object may be colliding with the other, I perform a point in concave object test. If all these tests pass, I find the closest face to the point and use that to resolve the collision.

### 4.3.2 Resolution

Soft body resolution loops through every point/face pair from the narrow phase. It will move the point and face in opposite directions by half the penetration depth to place the point on the surface of the face. I then calculate the velocity of the face at the point of collision, and calculate the change in velocities caused by the collisions. For soft body objects there is neither an angular velocity nor an inertia tensor, but this bounce must be calculated for every point and plane pair. The spring and pressure forces will act to restore each object's shape after the collision.

## 4.4 Impulse Based Physics

Impulse based collision resolution requires the positions of the two objects colliding ( $p_0$  and  $p_1$ ), the point of contact  $c$ , their respective velocities ( $v_0$  and  $v_1$ ), their inertia tensors ( $I_0$  and  $I_1$ ), their masses ( $m_0$  and  $m_1$ ), the normal of the collision  $\hat{n}$ , and the coefficient of restitution  $e$ . Using these, I calculate their relative velocity  $v_{01} = v_0 - v_1$  as well as the vectors from the center of each object to the point of collision represented by  $r_0 = c - p_0$  and  $r_1 = c - p_1$ . Using these equations, the impulse along the normal is computed as  $j = \frac{-(e+1)v_{01} \cdot \hat{n}}{\hat{n}^2 \left( \frac{1}{m_0} + \frac{1}{m_1} \right) + \hat{n} \cdot (I_0^{-1} \cdot (r_0 \times \hat{n}) \times r_0 + I_1^{-1} \cdot (r_1 \times \hat{n}) \times r_1)}$ . The angular and linear velocities are then increased as  $v'_0 = v_0 + \frac{j \cdot \hat{n}}{m_0}$ ,  $v'_1 = v_1 + \frac{j \cdot \hat{n}}{m_1}$ ,  $\omega'_0 = \omega_0 + I_0^{-1}(r_0 \times j\hat{n})$ , and  $\omega'_1 = \omega_1 + I_1^{-1}(r_1 \times j\hat{n})$ .

## 4.5 Separating Axis Theorem (SAT)

Separating Axis Theorem states that if two objects are colliding then there does not exist a single plane between these objects that does not intersect either object. Furthermore, if the objects are convex then there exists a plane between these objects that does not intersect either object.

For my SAT test, I use the normals from each object's collection of triangles. I project all the points of both objects onto these normals and check if any individual point of one object lies between at least two points of another object. I do not exclude parallel normals in my particular implementation since this test takes up a relatively minute amount of calculations compared to many other tests and algorithms I implement, and it also allows it to process more diverse sets of data.

The SAT returns true if it does not find a separating axis. If it fails to find a separating axis the algorithm may optionally locate the shallowest penetration depth for use with rigid body collisions.

## 4.6 Point in Concave Object

By the Jordan Curve Theorem, a point can be said to lie within a concave object if a ray projected from this point in any direction intersects with an odd number of triangles. Similarly, if the ray intersects an even number of triangles then it can be said that the point lies outside of the object.

The ray intersects triangle test first determines the point where the ray may intersect the triangle. Assume that we're given point  $p$ , ray direction  $\hat{r}$ , triangle vertices  $a$ ,  $b$ , and  $c$ , and the triangle's normal  $\hat{n}$ . A point along the ray is defined as  $q = p + t\hat{r}$ . The point lying on a triangle's plane is at  $t = \frac{(a-p) \cdot \hat{n}}{\hat{r} \cdot \hat{n}}$ . If this  $t$  is less than 0, then the ray points away from the triangle and does not intersect it. Otherwise, I perform a point in triangle test using the  $q$  I just computed. To optimize my implementation, I always assume  $\hat{r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ , so the calculation for  $t$  becomes  $t = \frac{a_z - p_z}{\hat{n}_z}$  and  $q$  is simplified to  $q = \begin{pmatrix} p_x \\ p_y \\ p_z + t \end{pmatrix}$ .

## 4.7 Point in Convex Object

The point in convex object test loops through every face of an object and tests which side of the face a point lies. If the point lies outside of any face then the point is outside of the object. Optionally, if the point is inside the object, this test returns the face it's closest to for additional computations outside of this algorithm.

## 4.8 Point in Triangle Test

Given the point  $p$  to be tested, the points  $t_0, t_1$ , and  $t_2$  defining a counter-clockwise triangle, and the triangle normal  $\hat{n}$ , one can calculate the values:

$$a = t_0 - p$$

$$u = b \times c$$

$$b = t_1 - p$$

$$v = c \times a$$

$$c = t_2 - p$$

$$w = a \times b$$

Which, in turn, determine if a point does not lie within a triangle if either  $u \cdot v < 0$  or  $u \cdot w < 0$  are true.

# 5 Updates

## 5.1 Soft Body Updates

Soft body objects have their velocities updated via a Runge-Kutta 4 (RK4) interpolator and their positions updated using a simple Euler integration method.

Soft body objects have all their points updated at once. Since the velocities are calculated using RK4 and every point in the object must be updated at the same time I have a special data structure that holds all the velocity estimates for RK4. Being RK4, I take four estimates, one at the beginning of the timestep, two at the halfway mark, and a final estimation at the end of the timestep. This returns the four velocities  $v_0, v_1, v_2$ , and  $v_3$ . The final velocity for each point is then calculated as  $v' = v + \frac{v_0 + v_3 + 2(v_1 + v_2)}{6}$ . The position is then simply calculated as  $p' = p + tv'$  with  $t$  being the total timestep of the frame.

Each RK4 estimation has three parts. First, all the velocities must be increased by the acceleration of the previous estimation via the equation  $v' = v + at$ . Then, all the points must be moved by the updated velocity using the equation  $p' = p + tv$  so that the pressure and spring forces may be correctly calculated. Finally, the total acceleration is calculated as  $a = \sum_i \frac{F_i}{m}$  with  $m$  being the mass of the particle and the  $F_i$ 's being each individual force applied to the particle (including gravity, pressure, and the spring forces).

## 5.2 Rigid Body Updates

Rigid body objects do not need the same amount of accuracy to remain stable, so I use a simple Euler-Cromer integrator. First linear and angular acceleration are calculated from the sum of all the



forces and torques respectively. Then, these are used to update the linear and angular velocities, which are in turn used to calculate the current position and angle of the object.

## 6 Known Issues

The rigid body simulation is extremely simple. In order to make it behave in a believable manner, I constrained the maximum angular and linear velocity. Additionally, the angular velocity has a minimum speed that when reached causes the angular velocity to go to zero and causes the angle to get locked to the nearest 90 degree increment when in contact with the ground.

The soft body object collision resolution has a minor glitch. Since the collision detection operates individually on vertices of an object and does not take into account edge collisions, the point may clip in such a way that the edge is stretched through the object. There are two common situations where this will happen, the first being if one of the objects has an unreasonably high velocity (which, if this happens, the soft body simulation is more likely to cause a noticeable break with overcompensating spring forces). The other possible scenario is if the spring and pressure forces are too low causing the soft body object to deflate (and possibly interpenetrate). Although this second scenario is more common, when it occurs the simulation is usually already in an inaccurate state since the object will be folded in on itself.

The soft body interactions will slow down the simulation noticeably if there are a large number of possibly intersecting points. With my current collision detection and resolution scheme this is unavoidable, but if one of the two tests (objects colliding or closest triangle) can be reduced by an order of magnitude this issue can be resolved. I had come across some papers utilizing voronoi regions, but did not have the time to implement them. I have also found references involving pixel shaders to perform the point in object test which would not cut down the number of operations, but it would be able to parallelize all the tests hopefully speeding up the process given adequate hardware.

## 7 Future Research

I would like to improve the efficiency of the soft body collision tests in order to make soft body simulations more accessible to game programmers. Additionally, I would hope to improve the visual accuracy of the simulation by implementing a self collision detection scheme.

Other soft body behaviors that would be worthwhile investigating and implementing would be tearing and joining of objects and implementing a simulation of density to allow rigid objects to float in and remain suspended within a soft body object. Although suspending objects inside a soft body object would present various difficulties, a point in concave polygon test I had previously implemented (and left in the appendix as *Point in Tetrahedron*) generated five determinates that can be used to calculate the barycentric coordinates of an object inside a tetrahedron and move it somewhat accurately when the parent soft body objects moves or wobbles.

## 8 Works Cited

Ericson, C. (2005). *Real-Time Collision Detection*. San Francisco, CA: Morgan Kaufmann Publishers.

Matyka, M., & Ollila, M. (2003, November 20). *Pressure Model of Soft Body Simulation*. Retrieved June 18, 2008, from Linköping University Electronic Press: <http://www.ep.liu.se/ecp/010/007/ecp01007.pdf>

Zhang, C., & Chen, T. (2001, May 18). *Efficient Feature Extraction for 2D/3D Objects in Mesh Representation*. Retrieved June 19, 2008, from Advanced Multimedia Processing Lab: [http://amp.ece.cmu.edu/Publication/Cha/icip01\\_Chapter.pdf](http://amp.ece.cmu.edu/Publication/Cha/icip01_Chapter.pdf)

## 9 Screen Captures

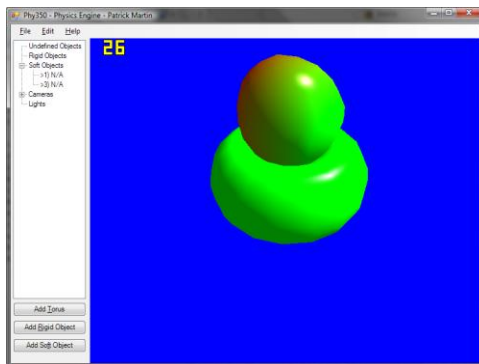


Figure 1 – A sphere resting on a torus using soft body dynamics.

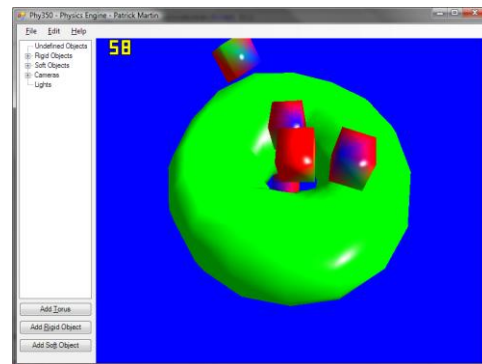


Figure 3 – Several rigid bodies interacting with a soft body torus.

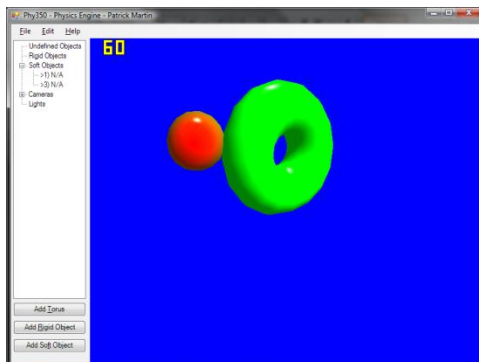


Figure 2 – A torus balancing and rolling on end while receiving a constant linear force.

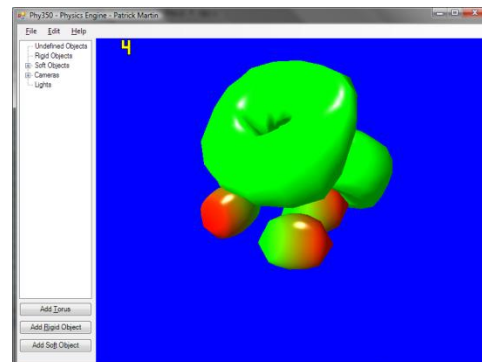


Figure 4 – Several soft body objects stacked and in contact.

## 10 Appendix A: Equations

### 10.1 Cross Product

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$$

### 10.2 Determinate

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{vmatrix} = a \begin{vmatrix} f & g & h \\ j & k & l \\ n & o & p \end{vmatrix} - b \begin{vmatrix} e & g & h \\ i & k & l \\ m & o & p \end{vmatrix} + c \begin{vmatrix} e & f & h \\ i & j & l \\ m & n & p \end{vmatrix} - d \begin{vmatrix} e & f & g \\ i & j & k \\ m & n & o \end{vmatrix}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} d & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & g \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

### 10.3 Dot Product

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \cdot \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x b_x + a_y b_y + a_z b_z$$

### 10.4 Euler-Cromer

$$v' = v + at$$

$$x' = x + at$$

- Acceleration:  $a$
- Velocity:  $v$
- Position:  $x$
- Timestep:  $t$

### 10.5 Hooke's Law

$$F_s = -k(x - x_0)$$

- Spring Coefficient:  $k$
- Spring Length:  $x$
- Spring Resting Length:  $x_0$

### 10.6 Ideal Gas Law

$$P = V^{-1}nRT$$

- Volume:  $V$
- Molar Count:  $n$
- Ideal Gas Constant:  $R$  (8.314)
- Absolute Temperature:  $T$

## 10.7 Impulse

$$v_{01} = v_0 - v_1$$

$$r_0 = c - p_0$$

$$r_1 = c - p_1$$

$$j = \frac{-(e + 1)v_{01} \cdot \hat{n}}{\hat{n}^2 \left( \frac{1}{m_0} + \frac{1}{m_1} \right) + \hat{n} \cdot (I_0^{-1} \cdot (r_0 \times \hat{n}) \times r_0 + I_1^{-1} \cdot (r_1 \times \hat{n}) \times r_1)}$$

$$v'_0 = v_0 + \frac{j \cdot \hat{n}}{m_0}$$

$$v'_1 = v_1 + \frac{j \cdot \hat{n}}{m_1}$$

$$\omega'_0 = \omega_0 + I_0^{-1}(r_0 \times j\hat{n})$$

$$\omega'_1 = \omega_1 + I_1^{-1}(r_1 \times j\hat{n})$$

- Velocities:  $v_0$   $v_1$
- Object Positions:  $p_0$   $p_1$
- Inertia Tensors:  $I_0$   $I_1$
- Collision Point:  $c$
- Collision Normal:  $\hat{n}$

## 10.8 Inertia Tensor of a Rectangular Prism

$$I = \begin{pmatrix} \frac{m}{12}(h^2 + d^2) & 0 & 0 \\ 0 & \frac{m}{12}(w^2 + d^2) & 0 \\ 0 & 0 & \frac{m}{12}(w^2 + h^2) \end{pmatrix}$$

- Mass:  $m$
- Width:  $w$
- Height:  $h$
- Depth:  $d$

## 10.9 Inverse of Diagonal Matrix

$$I^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & 0 \\ 0 & \frac{1}{b} & 0 \\ 0 & 0 & \frac{1}{c} \end{pmatrix}$$

- Diagonal Matrix:  $I = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$

## 10.10 Point in Tetrahedron

The following determinates must be the same size for the tetrahedron defined by  $a, b, c, o$  and the point  $p$ :

$$\begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ o_x & o_y & o_z & 1 \end{vmatrix} \begin{vmatrix} p_x & p_y & p_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ o_x & o_y & o_z & 1 \end{vmatrix} \begin{vmatrix} a_x & a_y & a_z & 1 \\ p_x & p_y & p_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \end{vmatrix} \begin{vmatrix} a_x & a_y & a_z & 1 \\ p_x & p_y & p_z & 1 \\ c_x & c_y & c_z & 1 \\ o_x & o_y & o_z & 1 \end{vmatrix} \begin{vmatrix} a_x & a_y & a_z & 1 \\ p_x & p_y & p_z & 1 \\ o_x & o_y & o_z & 1 \\ b_x & b_y & b_z & 1 \end{vmatrix}$$

## 10.11 Pressure Force

$$\vec{F}_p = P\hat{n}A$$

- Pressure:  $P$
- Normal:  $\hat{n}$
- Surface Area:  $A$

## 10.12 Runge-Kutta 4

$$v_0 = v(0)$$

$$v_1 = v\left(\frac{t}{2}\right)$$

$$v_2 = v\left(\frac{t}{2}\right)$$

$$v_3 = v(t)$$

$$v' = \frac{v_0 + v_3 + 2(v_1 + v_2)}{6}$$

- Velocity as a function of time:  $v(t)$
- Timestep:  $t$

### 10.13 Torque from Linear Force

$$\tau = r \times F$$

- Vector to particle:  $r$
- Linear force:  $F$