SIMULATING BLOOD FLOW IN MAYA: NPARTICLES AND FIELDS FOR DYNAMIC FLOW SIMULATION

Creating a volume axis field to influence nParticles

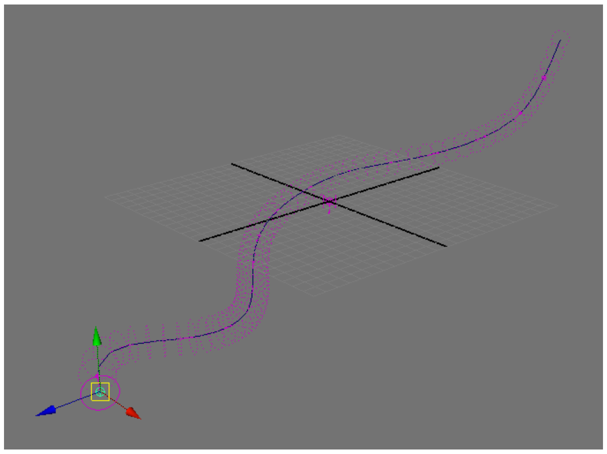
Open Maya (must be Maya 2013 for Mac; we will be using both Mental Ray and Instancer)

Create a CV curve in the shape of the path you want your blood cells to follow (for this demo you want it to be fairly long)

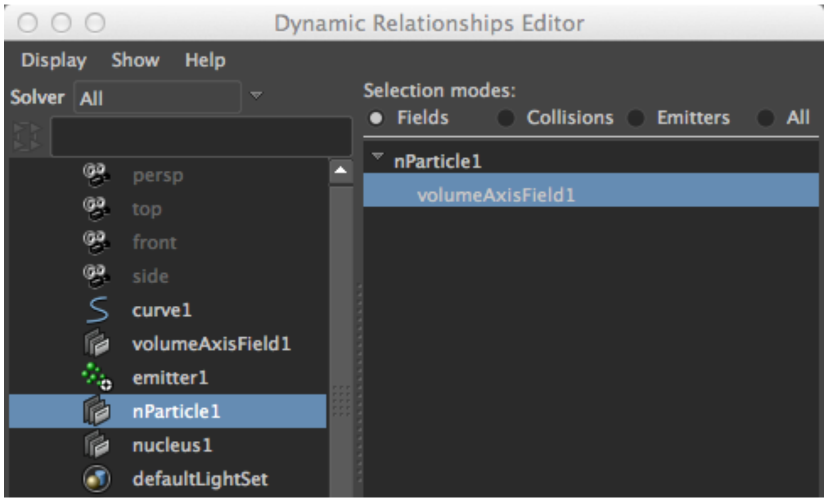
With that curve selected, go to Fields → Volume Curve; dotted circles will appear that represent the volume that the field controls; the centre of the volume will be marked with arrows that represent the direction of flow

Create a ball emitter: nParticles → create nParticles → enable balls; nParticles → create nParticles → create emitter

Reposition the emitter to the start of the field



Connect the nParticles to the field: Window → Relationship Editor → Dynamic Relationships; select the nParticles, and then enable the Volume Axis Field



Extend your timeline to 2000 frames and set playback to play every frame, max real time (you can access the playback speed by RMB clicking the timeline)

With the Volume Axis Field selected, go to the attribute editor and update the following:
Volume Axis Field Attributes → Magnitude: 5.000
Volume Axis Field Attributes → Attenuation: 3.000

With the Emitter selected, go to the attribute editor and update the following:
Basic Emitter Attributes → Rate (Particles/Sec): 30.000

With the Volume Axis Field selected, go to the attribute editor and update the following:
Volume Control Attributes → Section Radius: 1.000
Volume Control Attributes → Trap Inside: 1.000
Volume Speed Attributes → Along Axis: 1.000

With the nParticleShape node selected, go to the attribute editor and update the following:
Dynamic Properties → Enable Ignore Solver Wind and Ignore Solver Gravity
Dynamic Properties → Drag: 0.010

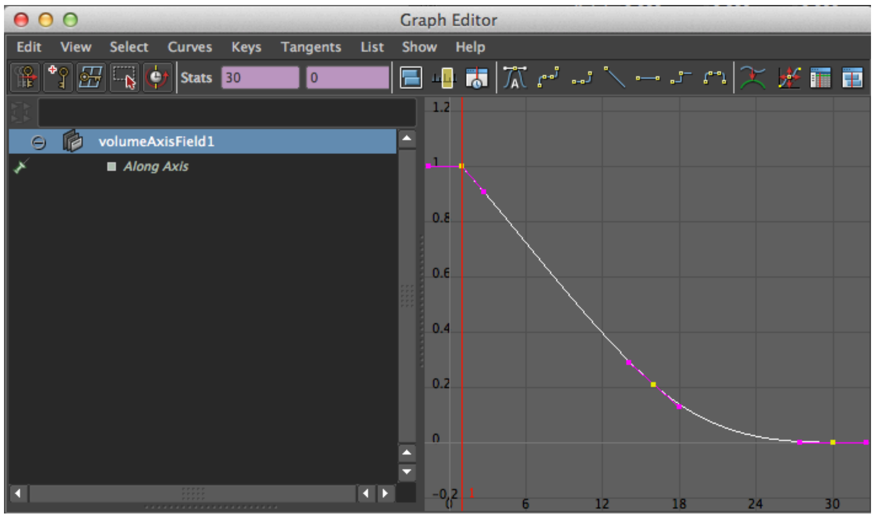
If you play back now, you should see that the nParticles that are closest to the radial centre of the field move more quickly than the ones towards the edges.

To see if you like your simulation, don't forget to render a playblast.

Keyframing the Along Axis Volume Speed to Achieve Pulsatile Flow

We can keyframe the Along Axis Volume Speed Attribute in the Volume Axis Field to make the particle motion pulsatile. Set a keyframe at t=1 frame at a value of 1, and another at t=30 frames at a value of 0.

Open Window → Animation Editor → Graph Editor and add one more keyframe midway between these two; set the t=1 keyframe to Linear Interpolation; drag the new midpoint keyframe so you get a shape that looks like this:



Still in the Graph Editor, set these keyframes to cycle via: Curves → Post-infinity → Cycle

To compensate for the fact that the Along Axis strength has decreased on average, let's update some attributes.

With the Volume Axis Field selected, go to the attribute editor and update the following:
Volume Axis Field Attributes → Magnitude: 8.000
Volume Axis Field Attributes → Attenuation: 0.300

Let's also increase the Drag on the nParticles to make them slow down between pulses. With the nParticleShape node selected, go to the attribute editor and update the following:
Dynamic Properties → Drag: 0.050

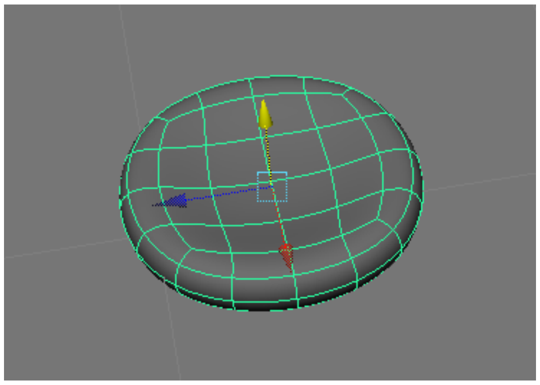
With the Emitter selected, go to the attribute editor and update the following:
Basic Emitter Attributes → Rate (Particles/Sec): 35.000

If you find that there are too many particles accumulating near the start of the field, you can decrease this value; if your field is looking too empty you can increase this value.

To see if you like your simulation, don't forget to render a playblast.

Using Instancer to Replace nParticles with Geometry that Dynamically Responds to Collisions

Create a piece of geometry that looks like a red blood cell by creating a non-polar sphere (smoothed polygon cube), scaling it to have a flattened shape, and pulling in the central vertices on each flattened side with soft selection enabled. You should end up with a flattened biconcave disc like this:



Let's rename it RBC and assign it a new red Blinn.

With RBC selected, go to nParticle → Instancer (Replacement) Options → check to make sure that RBC appears under the Instanced objects field, and hit Create

Play back a few frames so you can see some of your nParticles, and scale the RBC geometry so they fit just inside the nParticle balls; hide the original RBC geometry in the Outliner using the hotkey Control+H

With the nParticleShape node selected, go to the attributes and enable Object Display → Intermediate Object to hide the nParticles and show only the instanced geometry

Instead of creating random per particle rotations at creation, we can dynamically simulate the rotation of our RBCs in response to collisions: with the nParticleShape node still selected, go to Rotation → Enable Compute Rotation; Instancer (Geometry Replacement) → Rotation Options → Rotation: rotationPP

If you play back now, the nParticles will be spinning wildly near the emitter, but their motion will level out after a short distance away; you will be able to see your RBCs rotate in response to collisions

I increased the nParticleShape Attributes → Rotation → Rotation Damp to 0.010

To see if you like your simulation, don't forget to render a playblast.

If you are happy, you can also set an initial state with your vessels full of nParticles: with the nParticles selected, nSolver → Initial State → Set From Current

Using Instancer to Replace nParticles with Multiple Pieces of Geometry

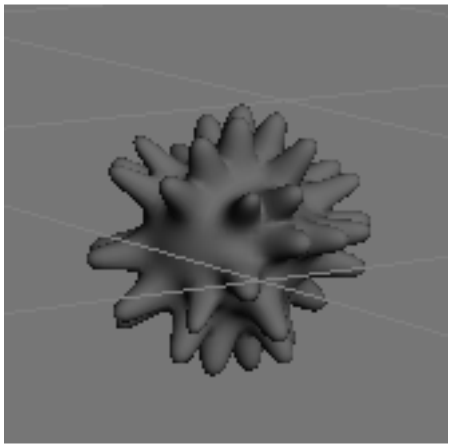
To include more than one piece of geometry in the Instancer (to include white blood cells, for instance), we can create an expression for the per particle Index value (IndexPP)

Create a polygon cube and smooth it to make a non-polar sphere; let's rename it WBC.

Select a few random faces on WBC: switch to face selection mode; Select → Select Using Constraints… → Constrain: Next Selection; Random → Enable Activate → Enter Ratio 0.5; marquee select over your cube

Using Edit Mesh → Extrude, extrude the selected faces, with Keep Faces Together disabled; taper the ends slightly

You should end up with something that looks like this:



Let's apply a new purple Blinn

Select WBC in the Outliner, then Command+Select the Instancer; in the Instancer attributes → Instancer → Instanced Objects → Add Selection

The Instancer will always select Object 0 unless we write an expression for IndexPP: nParticleShape attributes → Add Dynamic Attributes → General → create new attribute called Long Name: IndexPP → Data Type: Vector → Attribute Type: Per Particle (Array) → OK

Under nParticleShape attributes → Per Particle (Array) Attributes → RMB Index PP field → Creation Expression

This will open a new window for you to enter an expression to specify the distribution of instanced geometry.

```
int $i = rand(0,1001);
if($i > 1)
nParticleShape1.indexPP = 0;
if($i < 1)
nParticleShape1.indexPP=1;
```

Above, we first declared a dummy variable, integer $i, and assigned it a value between 0 and 1000; if $i is >999 it will be an RBC (remember that RBC is Object 0 in the Instancer) and if it is <1 it will be a WBC. This is in line with the true ratio of these cells in human blood.

In the nParticleShape attributes → Instancer (Geometry Replacement) → Enable Allow All Data Types → set Object Index: IndexPP

Play back enough frames that you spot a piece of WBC geometry; it may be necessary to scale your original to the approximate size of one of the nParticles; once you are happy, hide the original geometry in the Outliner by selecting it and using the hotkey Command+H

Using Passive Colliders for Branching Blood Vessel Networks

When you want to implement blood vessel branching, you will have to disable Trap Inside (set to 0 instead of 1) because otherwise, the second intersecting field won't be able to pull nParticles out of the first field.

To get around this, we can create geometry for the blood vessel wall and convert it into a passive collider.

Rebuild your CV curve that you used to make the first volume axis curve field so that its CVs are equally distributed: Edit Curves → Rebuild… → and bump up the number of spans. Let' s rename this CV Curve main.

Create a NURBS circle with 6 or 8 partitions.

With the CV curve selected, Shift+Select the NURBS circle and Surfaces → Extrude the circle over the curve. Scale the original circle until you have a vessel diameter you like, and the faces are roughly square in shape (you may also want to adjust the number of spans on your CV curve again at this point to make this happen).

With the NURBS surface selected, release the geometry to polygons (Edit → Modify → Convert NURBS to polygons); hide the NURBS surface in the outliner

Draw a second CV curve in the shape you want your branch to take. Rebuild this curve, also. Let's rename this CV Curve branch.

Select the face on the original blood vessel shape, and Shift+Select the branch CV curve, then go to Edit Mesh → Extrude

Check to make sure that all the normals point out (Display → Polygons → Display Face Normals); if the branches normals point in, make sure to flip them before moving on. I like to delete the end (cap) faces so that the particles can escape the tube to prevent them from backing up the upstream flow.

If you are happy with your geometry, make it into a Passive Collider: nMesh → Create Passive Collider

With the branch CV curve selected, create your next Volume Axis Curve field: Fields → Volume Curve (you will have to adjust the magnitude parameters, etc. to suit the smaller diameter)

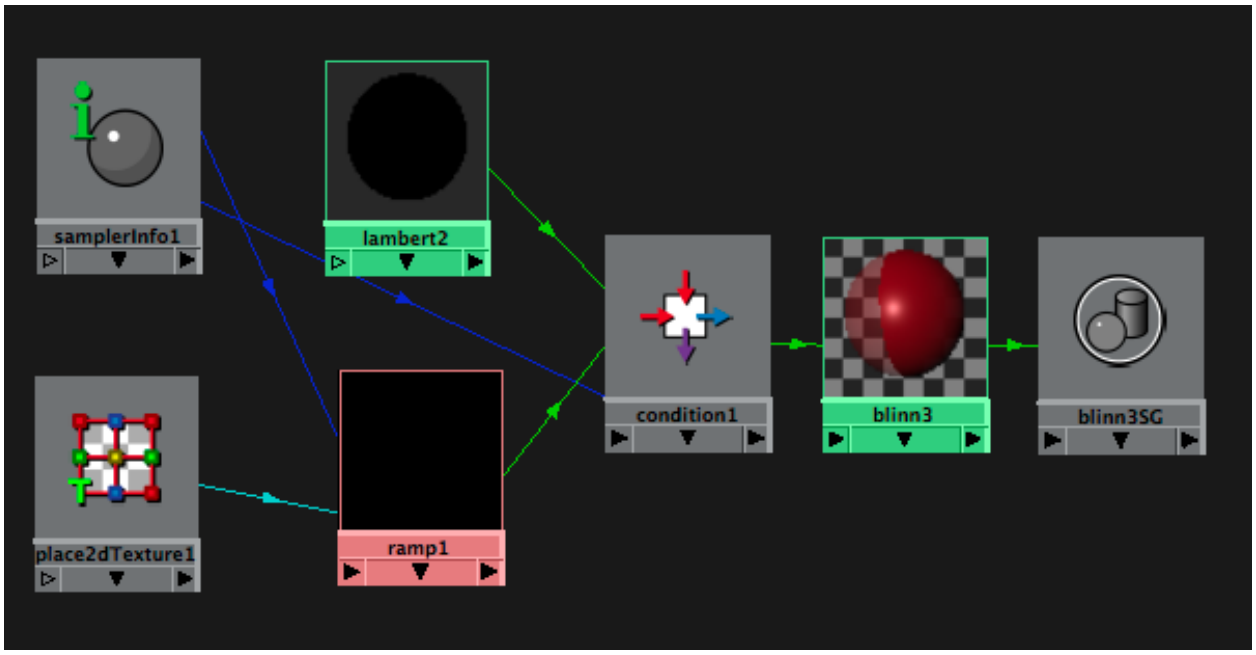
Apply the branch field to the nParticles by connecting them in the Dynamic Relationships Editor: Window → Relationship Editors → Dynamic Relationships

Now when you play back the simulation, once the main vessel fills, the nParticles should start to distribute into the branch. To check whether you like your results, don't forget to render a playblast.

Creating a Conditional Shader for Selective Transparency

For my animation, I wanted to show blood flow through a partially transparent front wall, but for the back wall to be opaque. To do this, I used a Sampler Info node (to assess whether the geometry's normal were facing towards or away from the camera, and to get the facing ratio for the areas facing towards the camera) and a condition node (to pipe the results of these assessments into the Transparency of a Blinn).

This is what my shading network looks like in the Hypershade:



Create a new red Blinn, a Condition node, a Ramp, and a Sampler Info node.

MMB drag the Sampler Info onto the Condition → Other… → connect the Flipped Normal to Condition 1: First Term

MMB drag the Sampler Info onto the Ramp → Other → connect the Facing Ratio to uvCoord: vCoord

Add a gradient of grey to black to the Ramp so that as the edges of your vessel turn away from the camera, they are more opaque (Fresnel effect)

MMB drag the Ramp onto the Condition → Other… → connect the outColor to colorIfTrue
MMB drag the black Lambert onto the Condition → Other… → connect the outColor to colorIfFalse

Double click to open the Condition Attributes menu and set the Second Term to 0, and the Operation to 0. If the Flipped Normal value returns a 0, the geometry's normal is facing towards the camera (ie. Is the front of the vessel), and the Condition node's outColor will be based on the Facing Ratio ramp. If the geometry's normal is facing away from the camera (ie. Is the back of the vessel), then the Condition's outColor will be black, based on the black Lambert.

Now we can MMB drag the Condition onto the red Blinn → Other… → connect the outColor to the Transparency.

I chose a red Blinn for the material because the specular highlight helps push back the blood cells inside it, so that they don't just look like they are passing in front of the vessel.

Play back a bit of your simulation so that there are blood cells in the frame, and then render out a frame so you can see what your material looks like rendered.