Cellular Automata - The Force of Color

Color is to be interpreted as force in these series of experiments. An intensive engine transfomring a planar surface into something other. This experiment is occurs in the space of Red, Green and Blue (RGB). The three values that make up a raster image can be thought of as vectors of force in a 3 dimensional space. This force changes over time according to a set of transformations based on Cellular Automata. This changing force is then applied to a flat plan, deforming it through time and creating a space of colored force.

Though the surfaces seem to become a jumbled mess, the structure of the image can still be seen in the form of the surface. By comparing the surfaces one can start to make out distinctions of quality. For example the first image has large areas of light and dark which tend to have high or low but roughly equal components of all three colors. This creates the stretched diagonal shape. The second image is largely monochromatic, being mostly a muddy dark brown though there are several small areas of blue. This creates the downward left displacement tendency of the surfaces. The slight variations in the colors also give the shape it more bulbous but undulating form comparted to the first image. Finally, the third image has distinct areas of red and blue. There are also bands of white at the top and bottom. These white bands create the wing like forms sprouting from the main form. The main form is split into two zones of the red and blue as each zone moved in slightly different directions.

The space of color is a fascinating one and seeing the structures of the images coming through in the interpretation of color as force is an interesting abstract study in the trasnference of information normally seen in one format but now converted into another.

■ Initialization, Includes and Front End

In[144]:=
```
<< Graphics`Animation`
<< Calculus`VectorAnalysis`
<< Graphics`PlotField3D`
```

In[147]:=
```
projectPath = "D:\\My Documents\\Columbia\\Algorithmic Morphologies\\"
```

Out[147]=
```
D:\My Documents\Columbia\Algorithmic Morphologies\
```

■ Tests

■ Differential Equations

In[148]:=

```
x[t_];
y[t_];
z[t_];
u[x_, y_, z_, t_] ;
v[x_, y_, z_, t_];
w[x_, y_, z_, t_];
ψ'[x_, y_, z_, t_];
```

In[25]:= $\psi'[x\_, y\_, z\_, t\_] == \frac{x'[t]}{u[x, y, z, t]} - \frac{y'[t]}{v[x, y, z, t]} - \frac{z'[t]}{w[x, y, z, t]}$

Out[25]= $\psi'[x\_, y\_, z\_, t\_] == \frac{x'[t]}{u[x, y, z, t]} - \frac{y'[t]}{v[x, y, z, t]} - \frac{z'[t]}{w[x, y, z, t]}$

The below statement just sets the variables "x", "y", and "z" to be functions of a variable "t". This parameterizes the functions, allowing it to iterate along a common variable.

```
x[t_];
y[t_];
z[t_];
```

The functions are defined as differential equations with respect to the parameter "t". These rates of change will need to be solved for the variables "x", "y", and "z".

```
fx[t_] = x'[t] == -(y[t] + z[t]);
fy[t_] = y'[t] == x[t] + a y[t];
fz[t_] = z'[t] == b + x[t] z[t] - c z[t];

sol1 = NDSolve[{fx1[t], fy1[t], fz1[t], x1[0] == x0, y1[0] == y0, z1[0] == z0},
  {x1, y1, z1}, {t, 0, 500}, MaxSteps → 100000]
```

```
{{x1 → InterpolatingFunction[{{0., 500.}}, <>],
  y1 → InterpolatingFunction[{{0., 500.}}, <>],
  z1 → InterpolatingFunction[{{0., 500.}}, <>]}}
```

```
ax[t_] = ax[0] +
```

■ Average Value Cellular Automata

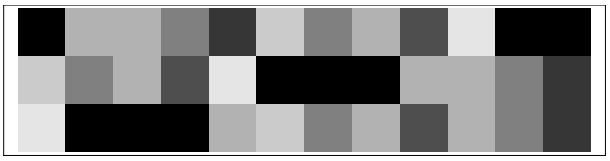
In[155]:=

```
fun[cells_List, t_Integer] :=
  (cells[[1]][[1]] + cells[[1]][[2]] + cells[[1]][[3]] + cells[[2]][[1]] +
     cells[[2]][[3]] + cells[[3]][[1]] + cells[[3]][[2]] + cells[[3]][[3]]) / 8;
```

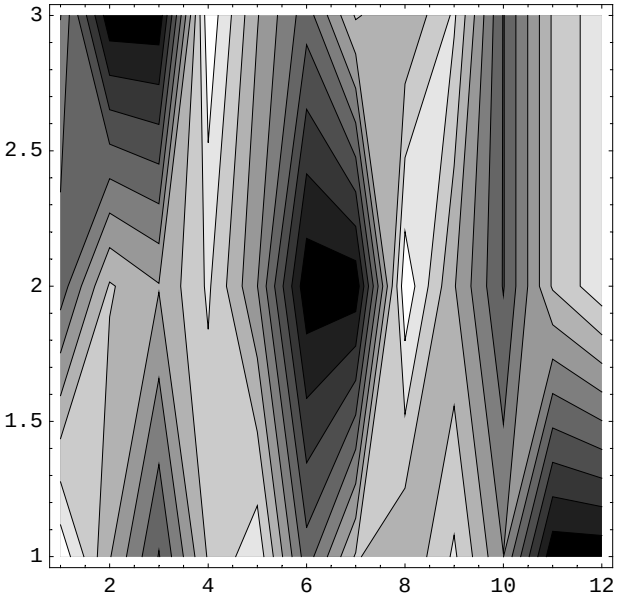
In[156]:=
```
ruleSet = {fun[#, #2] &, {}, {1, 1}} ;
initial = {{10., 3., -3., 5., 8., -2., 5., 3., 7., 1., -10., -10.},
   { -2., 5., 3., 7., 1., -10., -10., 10., 3., -3., 5., 8.},
   { 1., -10., -10., 10., 3., -2., 5., 3., 7., -3., 5., 8.}};
duration =
  50;
```

In[159]:=
```
cells01 = CellularAutomaton[ruleSet, initial, duration];
```

In[160]:=
```
ShowAnimation[Animate[ArrayPlot[cells01[[t]]], {t, 1, duration, 1}]];
```



In[161]:=
```
ShowAnimation[Animate[ContourGraphics[cells01[[t]]], {t, 1, duration, 1}]];
```



■ Cellular Automata Experiment: 01

■ Intialize

Defines System Name for file export later

In[32]:=
```
sys = "Cellular";
iteration = "01";
```

Import raster image, get RGB vector list and stripping the *Mathematica* Graphics data.

```
In[141]:=
picture = Import[projectPath <> "Initial_Pic-01.jpg"];
image = (picture[[1]][[1]] / 128) - 1;
Show[picture];
```

■ Acceleration Field

Wrap vectors in function named 'vector' so it will work with the Cellular Automata function. The Cellular Automata function utilizes a ListCorrelate while workings and the vector list causes errors. Later this 'vector' function will return the vector list unchanged.

```
In[36]:= imageSymb = Map[vector, image, {2}];
```

Acceleration

The Acceleration function takes a 2D list of symbols represeting 3D vectors and outputs a new symbol of the sum of the cross products of the vectors in the Von Neuman neighborhood times the time enveloped by a Sin curve. This will give an every increasing magintude to the acceleration that will increase in cycles.

```
In[37]:= Acceleration[cellList_List, t_Integer] :=
          Module[{a0 = ReleaseHold[cellList[[2]] [[2]]], da, a1, cells},
           cells = ReleaseHold[cellList];
           da = (CrossProduct[a0, cells[[1]][[2]]] + CrossProduct[a0, cells[[2]][[1]]] +
              CrossProduct[a0, cells[[2]][[3]]] + CrossProduct[a0, cells[[3]][[2]]]);
           a1 = a0 + da * t * Sin[t * (Pi / 8)]
          ]
```

Define the intial conditions of the Cellular Automata inculding the number of time steps.

```
In[38]:= ruleSet = {Acceleration[#1, #2] &, {}, {1, 1}};
         initial = imageSymb;
         duration = 3;
```

Run the Cellular Automat

```
In[41]:= AccelerationField =
           N[CellularAutomaton[ruleSet, initial, duration] /. vector[v_List] -> v];
```

■ Velocity Field

Velocity

Takes the acceleration field and a 2D list of 3D vectors as the initial velocity and compiles a velocity field at each time step by adding the acceleration at that time step to the previous time step's velocity.

```
In[42]:= Velocity[AField_List, Velocity0_List] := Module[{VField, V0},
           VField = {};
           V0 = Velocity0;
           For[k = 1, k <= Dimensions[AField][[1]], k++,
            V0 = AField[[k]] + V0;
            VField = Append[VField, V0]
           ];
           VField
          ]
```

Produce still initial velocity and calculate velocity field.

```
In[43]:= InitialVelocity = Table[0, {i, 1, Dimensions[AccelerationField][[2]]},
            {j, 1, Dimensions[AccelerationField][[3]]}];
         VelocityField = Velocity[AccelerationField, InitialVelocity];
```

■ Surfaces

SurfaceField

Similiar to the Velocity function. Given the velocity field and a 2D list of 3D vectors as the positions of the vertices of the surface, the function compiles a series of coordinates at each time step by moving each point by the velocity of that point at each time step.

```
In[45]:= SurfaceField[VField_List, Surface0_List] := Module[{XYZField},
           XYZField = {};
           XYZ0 = Surface0;
           For[k = 1, k <= Dimensions[VField][[1]], k++,
            XYZ0 = VField[[k]] + XYZ0;
            XYZField = Append[XYZField, XYZ0]
           ];
           XYZField
          ]
```

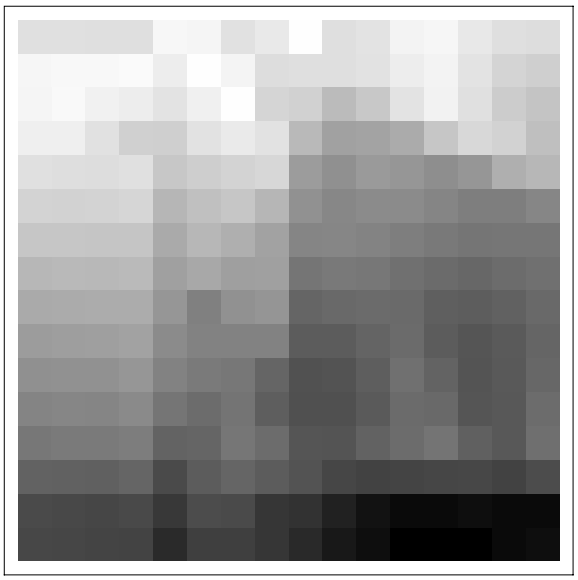
```
In[46]:= Plane = Table[{.25*i, .25*j, 0},
            {i, 1, Dimensions[VelocityField][[2]]}, {j, 1, Dimensions[VelocityField][[3]]}];
         TransformedSurfaces = SurfaceField[VelocityField, Plane];
```

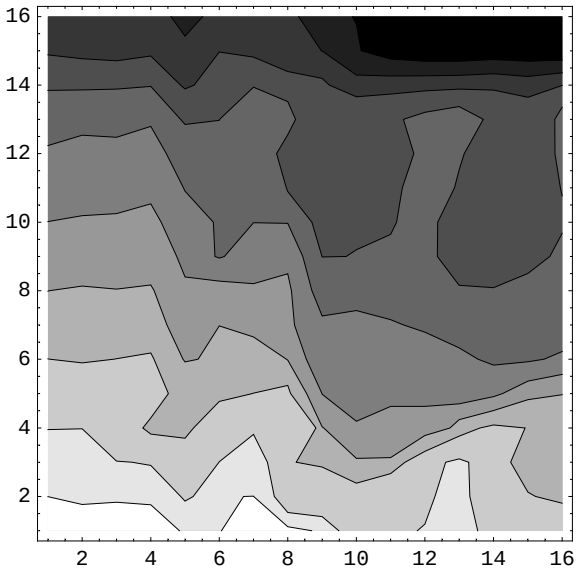
■ Visualization

```
In[53]:= PlotChannel[field_List, channel_Integer] :=
          ShowAnimation[
           Animate[GraphicsArray[{ArrayPlot[Table[field[[t]] [[i]] [[j]] [[channel]],
                {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]],
              ContourGraphics[Table[-field[[t]] [[i]] [[j]] [[channel]],
                {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]]}] ,
            {t, 1, Dimensions[field][[1]], 1}]]
```

Plots of X axis coordinates

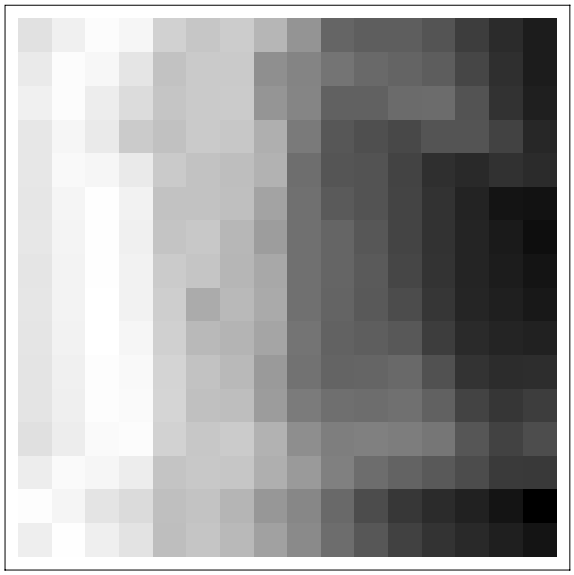
```
In[54]:= PlotChannel[TransformedSurfaces, 1]
```

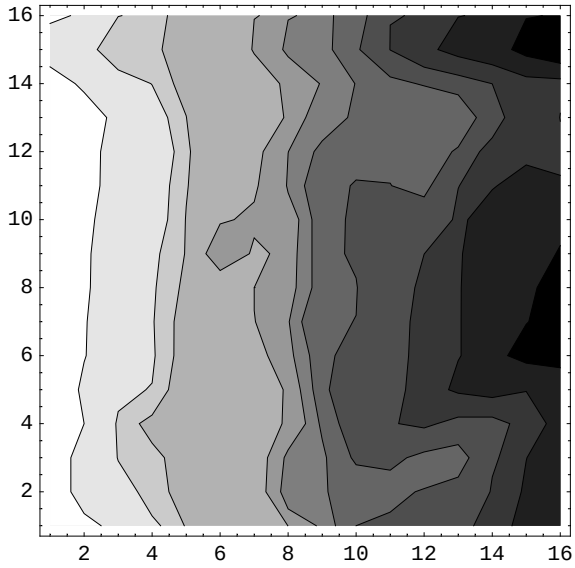




Plots of Y axis coordinates

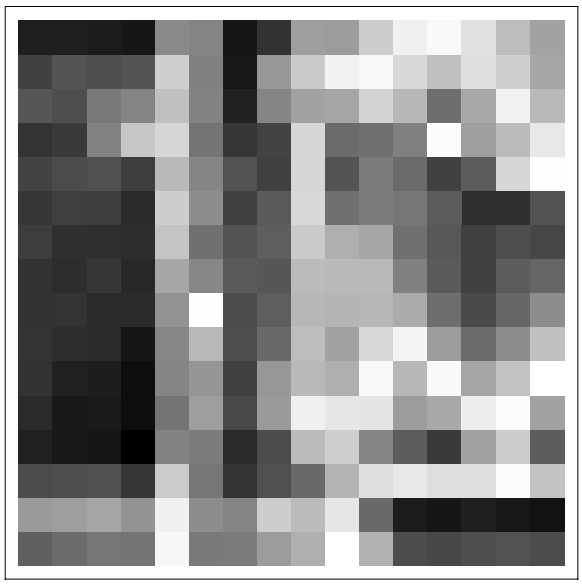
```
In[50]:= PlotChannel[TransformedSurfaces, 2]
```

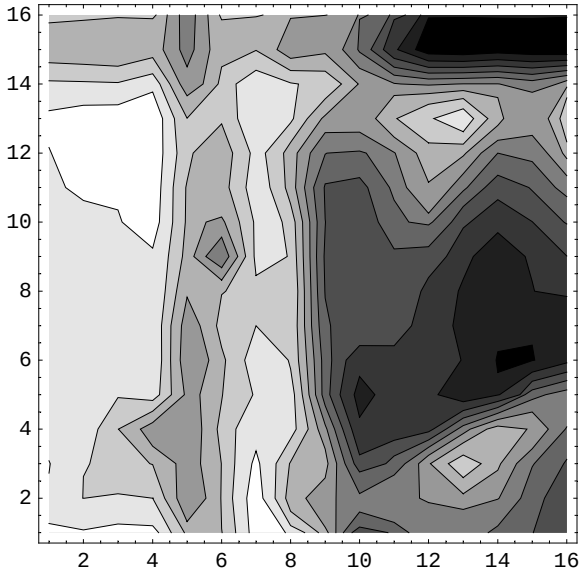




Plots of Z axis coordinates

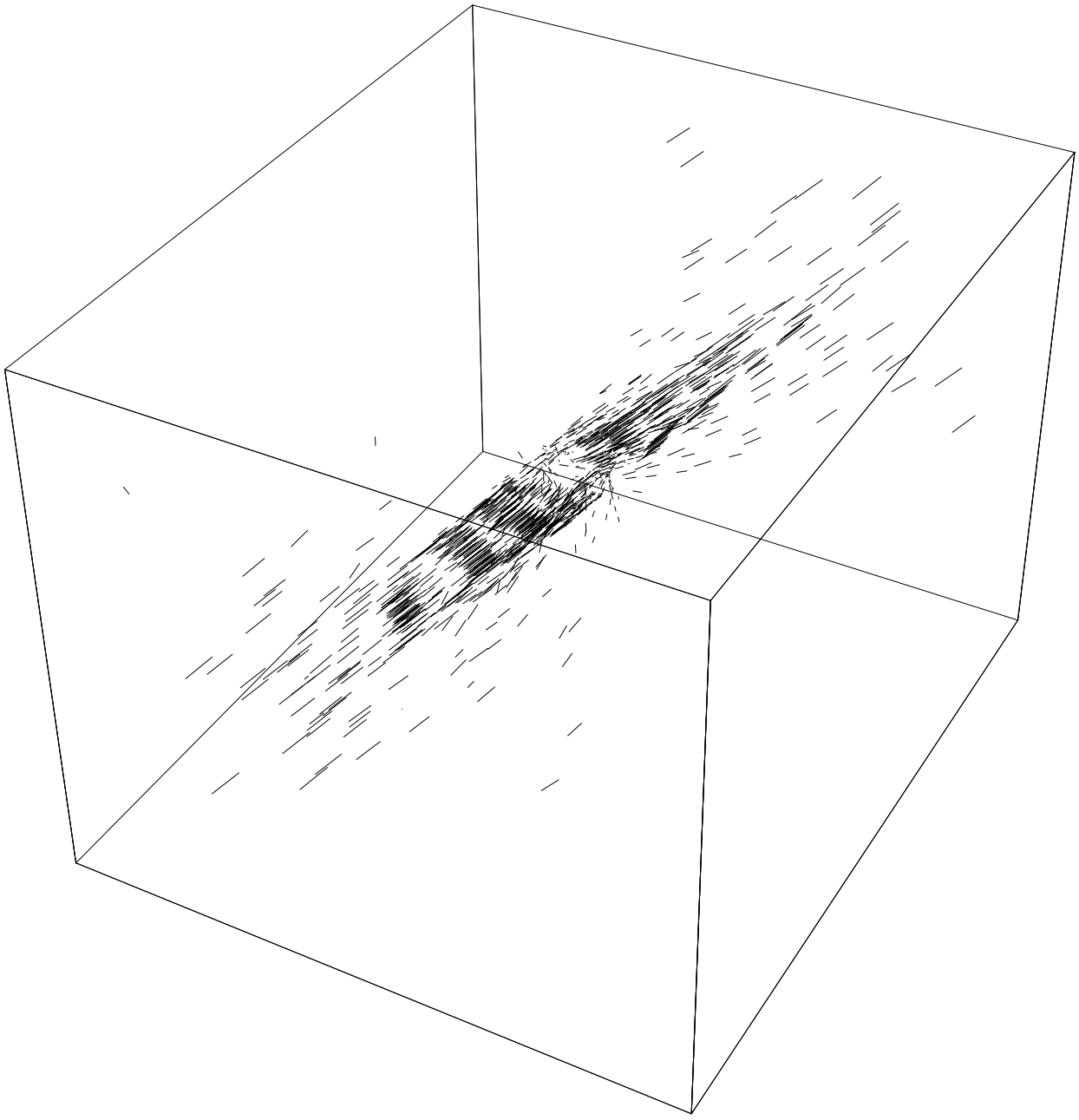
In[51]:= **PlotChannel[TransformedSurfaces, 3]**





Velocity vectors at points on transformed surfaces.

In[55]:=
```
FieldPlot =
 ListPlotVectorField3D[Flatten[Table[List[TransformedSurfaces[[t]][[i]][[j]],
     VelocityField[[1]][[i]][[j]]], {i, 1, Dimensions[TransformedSurfaces][[2]]},
    {j, 1, Dimensions[TransformedSurfaces][[3]]},
    {t, 1,  Dimensions[TransformedSurfaces][[1]]}], 2]]
```



Out[55]= - Graphics3D -

■ Output Surfaces

melEncoding

Requires a 2D List of 3D vectors and integer number for labeling purposes. Begin formatting array of vectors for export by converting to a string and replacing ','s with a space. Initialize counters. 'i' for CV's in u direction. 'j' for CV's in v direction. 'd' gives size of image. Then move through lists and replace characters with mel code to select the surface and move the CV according to the SurfaceArray vectors. Finally add mel code header that creates the nurbs suface. Return String of mel code that creates nurbsPlane and moves CV's to locations in 2D list.

```
In[57]:= melEncoding[surfaceList_List, imgNum_Integer] := Module[
           {i, j, d, SurfaceArray, SurfaceArrayReplaced, SurfaceArrayFinal, melHeader},
           i = 0;
           j = 0;
           d = Dimensions[surfaceList];
           SurfaceArray = ExportString[surfaceList, "Table"];
           SurfaceArrayReplaced = StringReplace[SurfaceArray, {", " -> " "}];
           SurfaceArrayFinal = SurfaceArrayReplaced;
           For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
            replacePositions = StringPosition[SurfaceArrayFinal, "{"];
            SurfaceArrayFinal = StringReplacePart[SurfaceArrayFinal, "select $surface" <>
               ExportString[imgNum, "Text"] <> "; move ", replacePositions[[1]]];
           ];
           For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
            replacePositions = StringPosition[SurfaceArrayFinal, "}"];
            SurfaceArrayFinal =
             StringReplacePart[SurfaceArrayFinal, " .cv[" <> ExportString[i, "Text"] <>
               "][" <> ExportString[j, "Text"] <> "]; ", replacePositions[[1]]];
            If[i < d[[1]] - 1, i++, i = 0; j++;];
           ];
           melSurfaceHeader = "\n$surface" <> ExportString[imgNum, "Text"] <>
             "= `nurbsPlane -ch on -o on -po 0 -ax 0 1 0 -u " <>
             ExportString[(d[[1]] - 3), "Text"] <> " -v " <>
             ExportString[(d[[2]] - 3), "Text"] <> " -w 150 -lr 1`;\n";
           melSurfaceHeader <> SurfaceArrayFinal
          ]
```

melSurfaceCompile

Takes the list of surface coordinates and produces one surface at each time step.

```
In[58]:= melSurfaceCompile[TransformedSurfaces_List] := Module[{melCode},
           melCode = "";
           For[n = 1, n ≤ Dimensions[TransformedSurfaces][[1]], n++,
             melCode = melCode <> melEncoding[N[TransformedSurfaces[[n]]], n];
            ]
            melCode
          ]

In[71]:= melCode = melSurfaceCompile[TransformedSurfaces];
```

Export strings into .mel file.

```
In[72]:= Export[projectPath <> sys <> iteration <> ".mel", melCode, "Text"]
Out[72]= D:\My Documents\Columbia\Algorithmic Morphologies\Cellular01.mel

In[56]:= Export[projectPath <> sys <> iteration <> "-01.jpg",
          FieldPlot, "JPEG", ImageSize → {800, 800}]
Out[56]= D:\My Documents\Columbia\Algorithmic Morphologies\Cellular01-01.jpg
```

■ Cellular Automata Experiment: 02

■ Intialize

Defines System Name for file export later

```
In[86]:= sys = "Cellular";
         iteration = "02";
```

Import raster image, get RGB vector list and stripping the *Mathematica* Graphics data.

```
In[138]:=
        picture = Import [projectPath <> "Initial_Pic-02.jpg"];
        image = (picture[[1]][[1]] / 128) - 1;
        Show[picture];
```

■ Acceleration Field

Wrap vectors in function named 'vector' so it will work with the Cellular Automata function. The Cellular Automata function utilizes a ListCorrelate while workings and the vector list causes errors. Later this 'vector' function will return the vector list unchanged.

```
In[63]:= imageSymb = Map[vector, image, {2}];
```

Acceleration

The Acceleration function takes a 2D list of symbols represeting 3D vectors and outputs a new symbol of the sum of the cross products of the vectors in the Von Neuman neighborhood times the time enveloped by a Sin curve. This will give an every increasing magintude to the acceleration that will increase in cycles.

```
In[64]:= Acceleration[cellList_List, t_Integer] :=
          Module[{a0 = ReleaseHold[cellList[[2]][[2]]], da, a1, cells},
           cells = ReleaseHold[cellList];
           da = (CrossProduct[a0, cells[[1]][[2]]] + CrossProduct[a0, cells[[2]][[1]]] +
              CrossProduct[a0, cells[[2]][[3]]] + CrossProduct[a0, cells[[3]][[2]]]);
           a1 = a0 + da * t * Sin[t * (Pi / 8)]
          ]
```

Define the intial conditions of the Cellular Automata inculding the number of time steps.

```
In[65]:= ruleSet = {Acceleration[#1, #2] &, {}, {1, 1}};
         initial = imageSymb;
         duration = 3;
```

Run the Cellular Automat

```
In[68]:= AccelerationField =
           N[CellularAutomaton[ruleSet, initial, duration] /. vector[v_List] -> v];
```

■ Velocity Field

Velocity

Takes the acceleration field and a 2D list of 3D vectors as the initial velocity and compiles a velocity field at each time step by adding the acceleration at that time step to the previous time step's velocity.

```
In[69]:= Velocity[AField_List, Velocity0_List] := Module[{VField, V0},
           VField = {};
           V0 = Velocity0;
           For[k = 1, k <= Dimensions[AField][[1]], k++,
            V0 = AField[[k]] + V0;
            VField = Append[VField, V0]
           ];
           VField
          ]
```

Produce still initial velocity and calculate velocity field.

```
In[70]:= InitialVelocity = Table[0, {i, 1, Dimensions[AccelerationField][[2]]},
            {j, 1, Dimensions[AccelerationField][[3]]}];
         VelocityField = Velocity[AccelerationField, InitialVelocity];
```

■ Surfaces

SurfaceField

Similiar to the Velocity function. Given the velocity field and a 2D list of 3D vectors as the positions of the vertices of the surface, the function compiles a series of coordinates at each time step by moving each point by the velocity of that point at each time step.

```
In[72]:= SurfaceField[VField_List, Surface0_List] := Module[ {XYZField},
           XYZField = {};
           XYZ0 = Surface0;
           For[k = 1, k <= Dimensions[VField][[1]], k++,
            XYZ0 = VField[[k]] + XYZ0;
            XYZField = Append[XYZField, XYZ0]
           ];
           XYZField
          ]
```

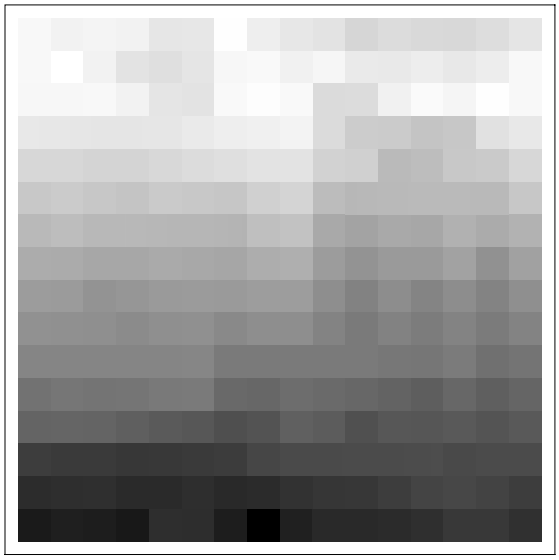
```
In[73]:= Plane = Table[{.25 * i, .25 * j, 0},
            {i, 1, Dimensions[VelocityField][[2]]}, {j, 1, Dimensions[VelocityField][[3]]}];
         TransformedSurfaces = SurfaceField[VelocityField, Plane];
```

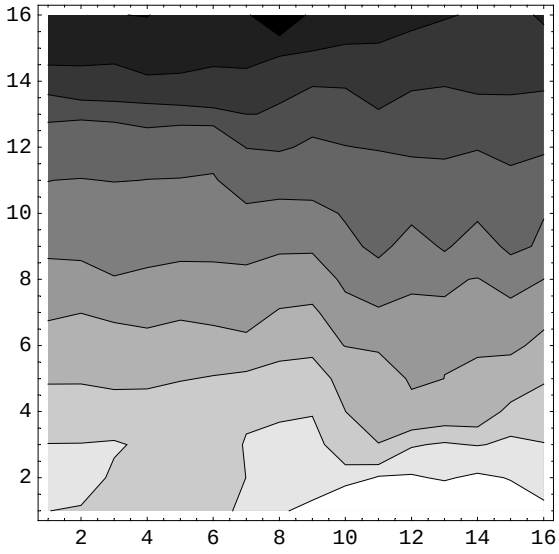
■ Visualization

```
In[75]:= PlotChannel[field_List, channel_Integer] :=
          ShowAnimation[
           Animate[GraphicsArray[{ArrayPlot[Table[field[[t]] [[i]] [[j]] [[channel]],
                {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]],
              ContourGraphics[Table[-field[[t]] [[i]] [[j]] [[channel]],
                {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]]}] ,
            {t, 1, Dimensions[field][[1]], 1}]]
```

Plots of X axis coordinates

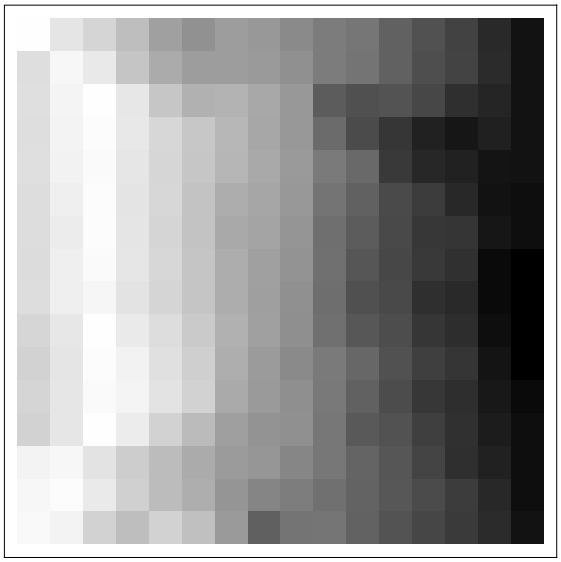
```
In[76]:= PlotChannel[TransformedSurfaces, 1]
```

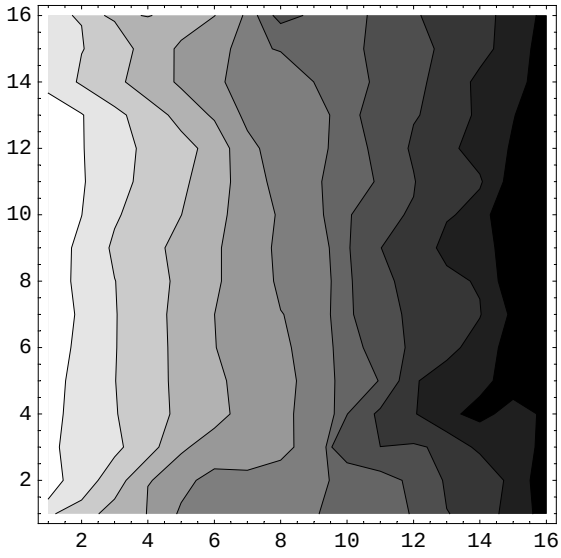




Plots of Y axis coordinates

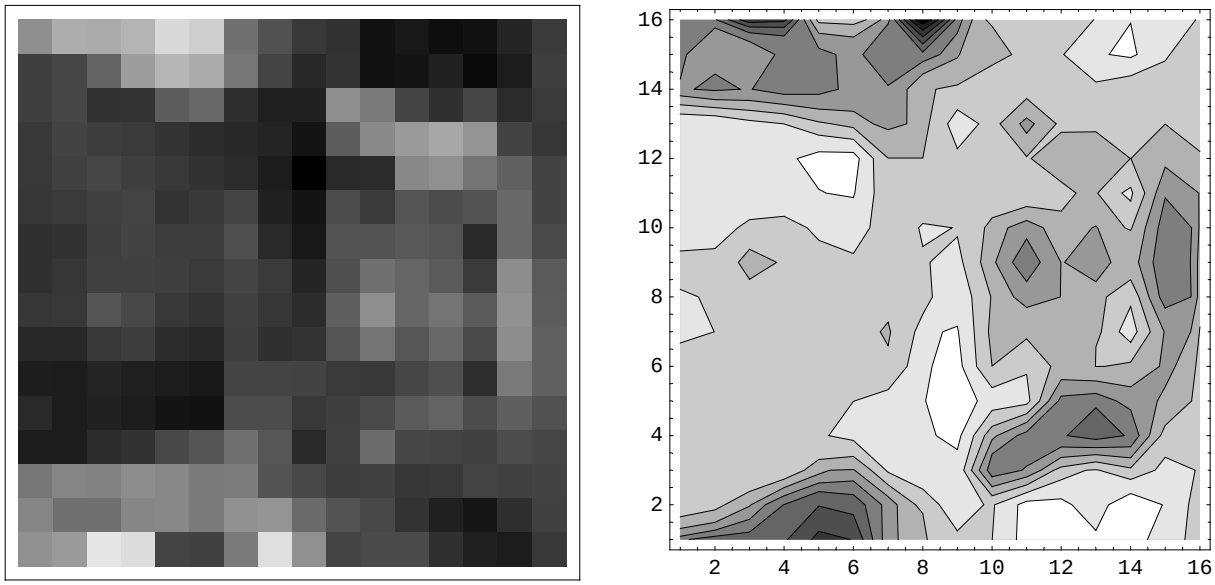
In[77]:= **PlotChannel[TransformedSurfaces, 2]**





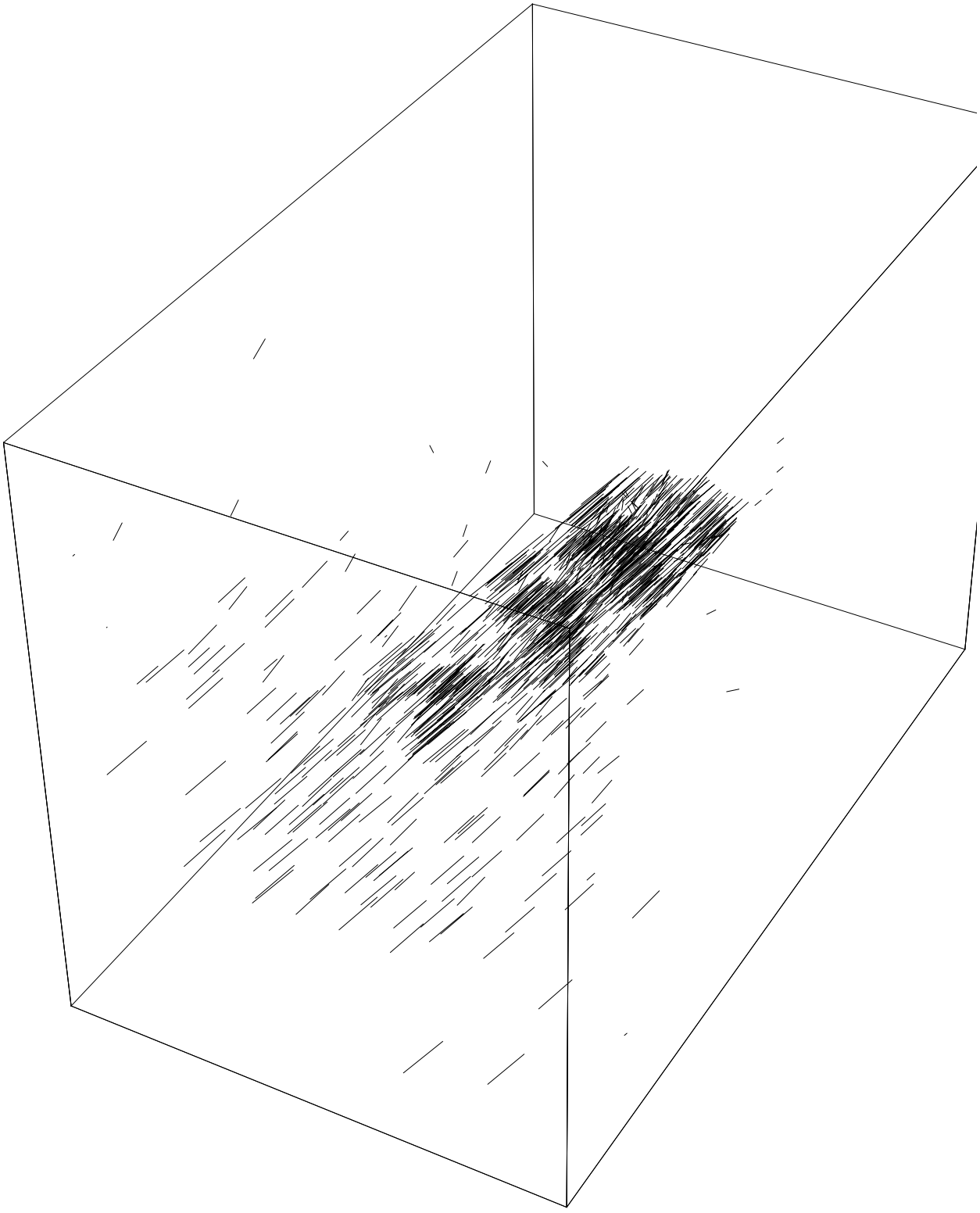
Plots of Z axis coordinates

In[78]:= **PlotChannel[TransformedSurfaces, 3]**



Velocity vectors at points on transformed surfaces.

```
In[89]:= FieldPlot =
          ListPlotVectorField3D[Flatten[Table[List[TransformedSurfaces[[t]][[i]][[j]],
              VelocityField[[1]][[i]][[j]]], {i, 1, Dimensions[TransformedSurfaces][[2]]},
            {j, 1, Dimensions[TransformedSurfaces][[3]]},
            {t, 1,  Dimensions[TransformedSurfaces][[1]]}], 2]]
```



Out[89]= -Graphics3D-

■ Output Surfaces

melEncoding

Requires a 2D List of 3D vectors and integer number for labeling purposes. Begin formatting array of vectors for export by converting to a string and replacing ','s with a space. Initialize counters. 'i' for CV's in u direction. 'j' for CV's in v direction. 'd' gives size of image. Then move through lists and replace characters with mel code to select the surface and move the CV according to the SurfaceArray vectors. Finally add mel code header that creates the nurbs suface. Return String of mel code that creates nurbsPlane and moves CV's to locations in 2D list.

```
In[80]:= melEncoding[surfaceList_List, imgNum_Integer] := Module[
           {i, j, d, SurfaceArray, SurfaceArrayReplaced, SurfaceArrayFinal, melHeader},
           i = 0;
           j = 0;
           d = Dimensions[surfaceList];
           SurfaceArray = ExportString[surfaceList, "Table"];
           SurfaceArrayReplaced = StringReplace[SurfaceArray, {", " -> " "}];
           SurfaceArrayFinal = SurfaceArrayReplaced;
           For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
            replacePositions = StringPosition[SurfaceArrayFinal, "{"];
            SurfaceArrayFinal = StringReplacePart[SurfaceArrayFinal, "select $surface" <>
               ExportString[imgNum, "Text"] <> "; move ", replacePositions[[1]]];
           ];
           For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
            replacePositions = StringPosition[SurfaceArrayFinal, "}"];
            SurfaceArrayFinal =
             StringReplacePart[SurfaceArrayFinal, " .cv[" <> ExportString[i, "Text"] <>
               "][" <> ExportString[j, "Text"] <> "]; ", replacePositions[[1]]];
            If[i < d[[1]] - 1, i++, i = 0; j++;];
           ];
           melSurfaceHeader = "\n$surface" <> ExportString[imgNum, "Text"] <>
             "= `nurbsPlane -ch on -o on -po 0 -ax 0 1 0 -u " <>
             ExportString[(d[[1]] - 3), "Text"] <> " -v " <>
             ExportString[(d[[2]] - 3), "Text"] <> " -w 150 -lr 1`;\n";
           melSurfaceHeader <> SurfaceArrayFinal
          ]
```

melSurfaceCompile

Takes the list of surface coordinates and produces one surface at each time step.

```
In[81]:= melSurfaceCompile[TransformedSurfaces_List] := Module[{melCode},
           melCode = "";
           For[n = 1, n ≤ Dimensions[TransformedSurfaces][[1]], n++,
             melCode = melCode <> melEncoding[N[TransformedSurfaces[[n]]], n];
            ]
            melCode
          ]

In[82]:= melCode = melSurfaceCompile[TransformedSurfaces];
```

Export strings into .mel file.

```
In[83]:= Export[projectPath <> sys <> iteration <> ".mel", melCode, "Text"]

Out[83]= D:\My Documents\Columbia\Algorithmic Morphologies\Cellular02.mel
```

```
In[90]:= Export[projectPath <> sys <> iteration <> "-01.jpg",
          FieldPlot, "JPEG", ImageSize → {800, 800}]

Out[90]= D:\My Documents\Columbia\Algorithmic Morphologies\Cellular02-01.jpg
```

■ Cellular Automata Experiment: 03

■ Intialize

Defines System Name for file export later

```
In[91]:= sys = "Cellular";
         iteration = "03";
```

Import raster image, get RGB vector list and stripping the *Mathematica* Graphics data.

```
In[135]:=
        picture = Import [projectPath <> "Initial_Pic-03.jpg"];
        image = (picture[[1]][[1]] / 128) - 1;
        Show[picture];
```

■ Acceleration Field

Wrap vectors in function named 'vector' so it will work with the Cellular Automata function. The Cellular Automata function utilizes a ListCorrelate while workings and the vector list causes errors. Later this 'vector' function will return the vector list unchanged.

```
In[95]:= imageSymb = Map[vector, image, {2}];
```

Acceleration

The Acceleration function takes a 2D list of symbols represeting 3D vectors and outputs a new symbol of the sum of the cross products of the vectors in the Von Neuman neighborhood times the time enveloped by a Sin curve. This will give an every increasing magintude to the acceleration that will increase in cycles.

```
In[96]:= Acceleration[cellList_List, t_Integer] :=
          Module[{a0 = ReleaseHold[cellList[[2]][[2]]], da, a1, cells},
           cells = ReleaseHold[cellList];
           da = (CrossProduct[a0, cells[[1]][[2]]] + CrossProduct[a0, cells[[2]][[1]]] +
              CrossProduct[a0, cells[[2]][[3]]] + CrossProduct[a0, cells[[3]][[2]]]);
           a1 = a0 + da * t * Sin[t * (Pi / 8)]
          ]
```

Define the intial conditions of the Cellular Automata inculding the number of time steps.

```
In[97]:= ruleSet = {Acceleration[#1, #2] &, {}, {1, 1}};
         initial = imageSymb;
         duration = 3;
```

Run the Cellular Automat

```
In[100]:=
        AccelerationField =
          N[CellularAutomaton[ruleSet, initial, duration] /. vector[v_List] -> v];
```

■ Velocity Field

Velocity

Takes the acceleration field and a 2D list of 3D vectors as the initial velocity and compiles a velocity field at each time step by adding the acceleration at that time step to the previous time step's velocity.

```
In[101]:=
        Velocity[AField_List, Velocity0_List] := Module[{VField, V0},
          VField = {};
          V0 = Velocity0;
          For[k = 1, k <= Dimensions[AField][[1]], k++,
           V0 = AField[[k]] + V0;
           VField = Append[VField, V0]
          ];
          VField
         ]
```

Produce still initial velocity and calculate velocity field.

```
In[102]:=
        InitialVelocity = Table[0, {i, 1, Dimensions[AccelerationField][[2]]},
           {j, 1, Dimensions[AccelerationField][[3]]}];
        VelocityField = Velocity[AccelerationField, InitialVelocity];
```

■ Surfaces

SurfaceField

Similiar to the Velocity function. Given the velocity field and a 2D list of 3D vectors as the positions of the vertices of the surface, the function compiles a series of coordinates at each time step by moving each point by the velocity of that point at each time step.

In[104]:=
```
SurfaceField[VField_List, Surface0_List] := Module[ {XYZField},
  XYZField = {};
  XYZ0 = Surface0;
  For[k = 1, k <= Dimensions[VField][[1]], k++,
   XYZ0 = VField[[k]] + XYZ0;
   XYZField = Append[XYZField, XYZ0]
  ];
  XYZField
 ]
```

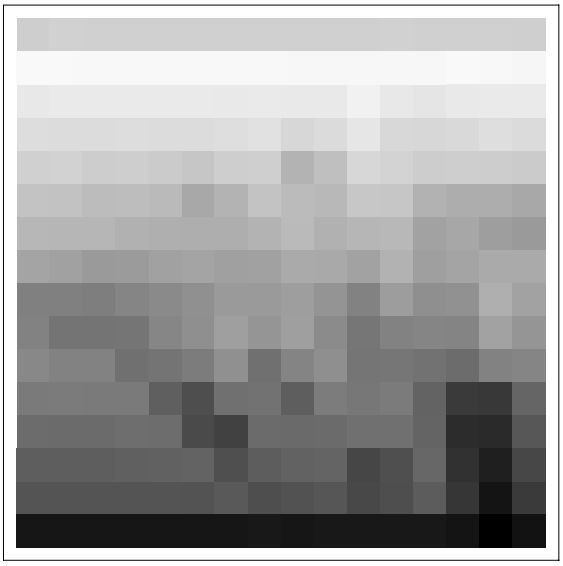
In[105]:=
```
Plane = Table[{.25 * i, .25 * j, 0},
   {i, 1, Dimensions[VelocityField][[2]]}, {j, 1, Dimensions[VelocityField][[3]]}];
TransformedSurfaces = SurfaceField[VelocityField, Plane];
```

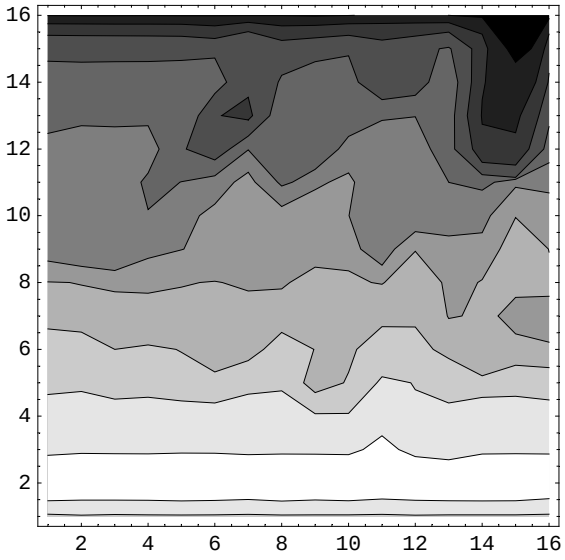
■ Visualization

In[107]:=
```
PlotChannel[field_List, channel_Integer] :=
 ShowAnimation[
  Animate[GraphicsArray[{ArrayPlot[Table[field[[t]] [[i]] [[j]] [[channel]],
       {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]],
     ContourGraphics[Table[-field[[t]] [[i]] [[j]] [[channel]],
       {i, 1, Dimensions[field][[2]]}, {j, 1, Dimensions[field][[3]]}]]}] ,
   {t, 1, Dimensions[field][[1]], 1}]]
```

Plots of X axis coordinates

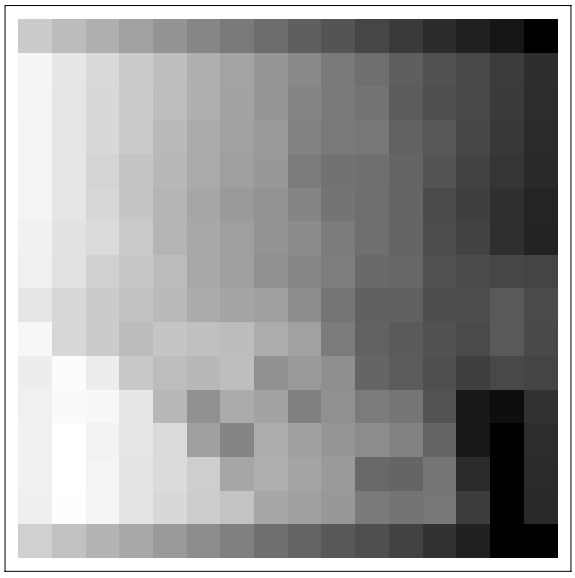
```
PlotChannel[TransformedSurfaces, 1];
```

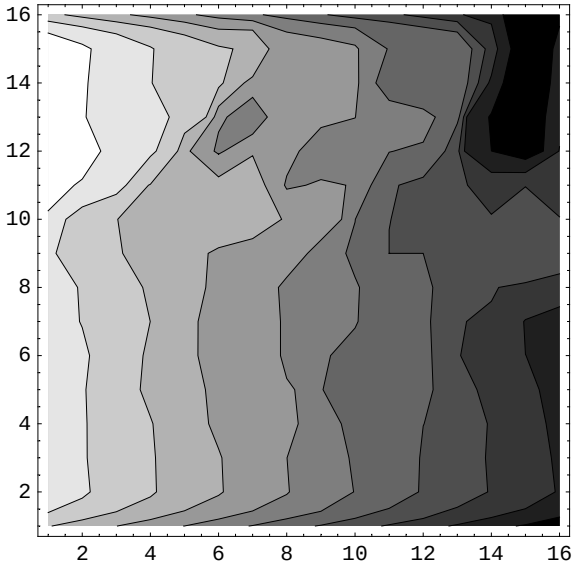




Plots of Y axis coordinates

```
PlotChannel[TransformedSurfaces, 2];
```

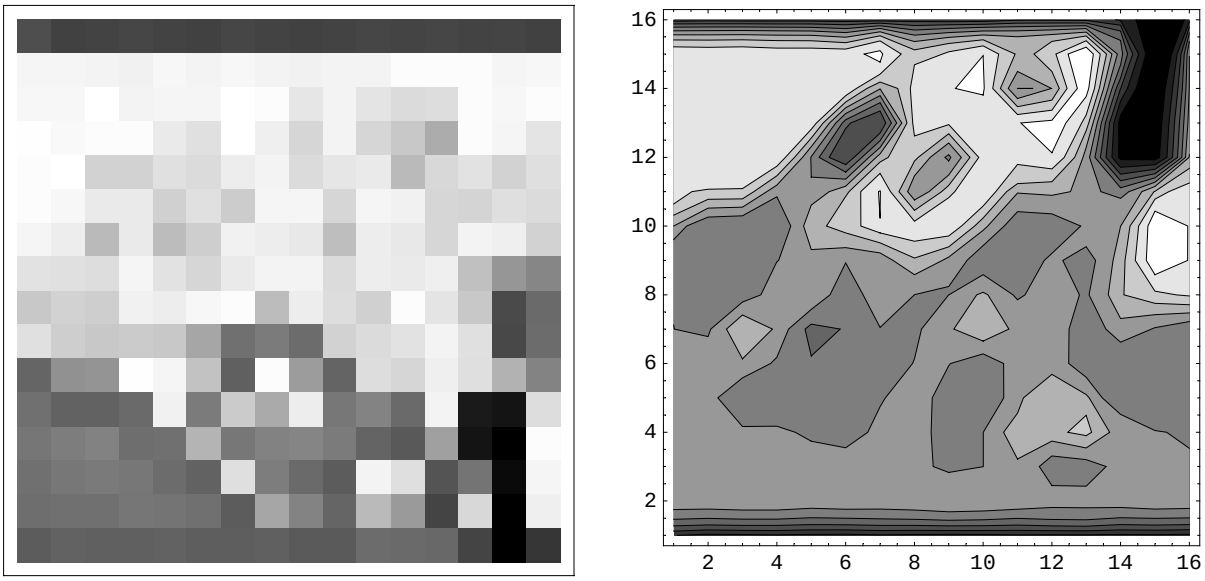




Plots of Z axis coordinates

In[110]:=

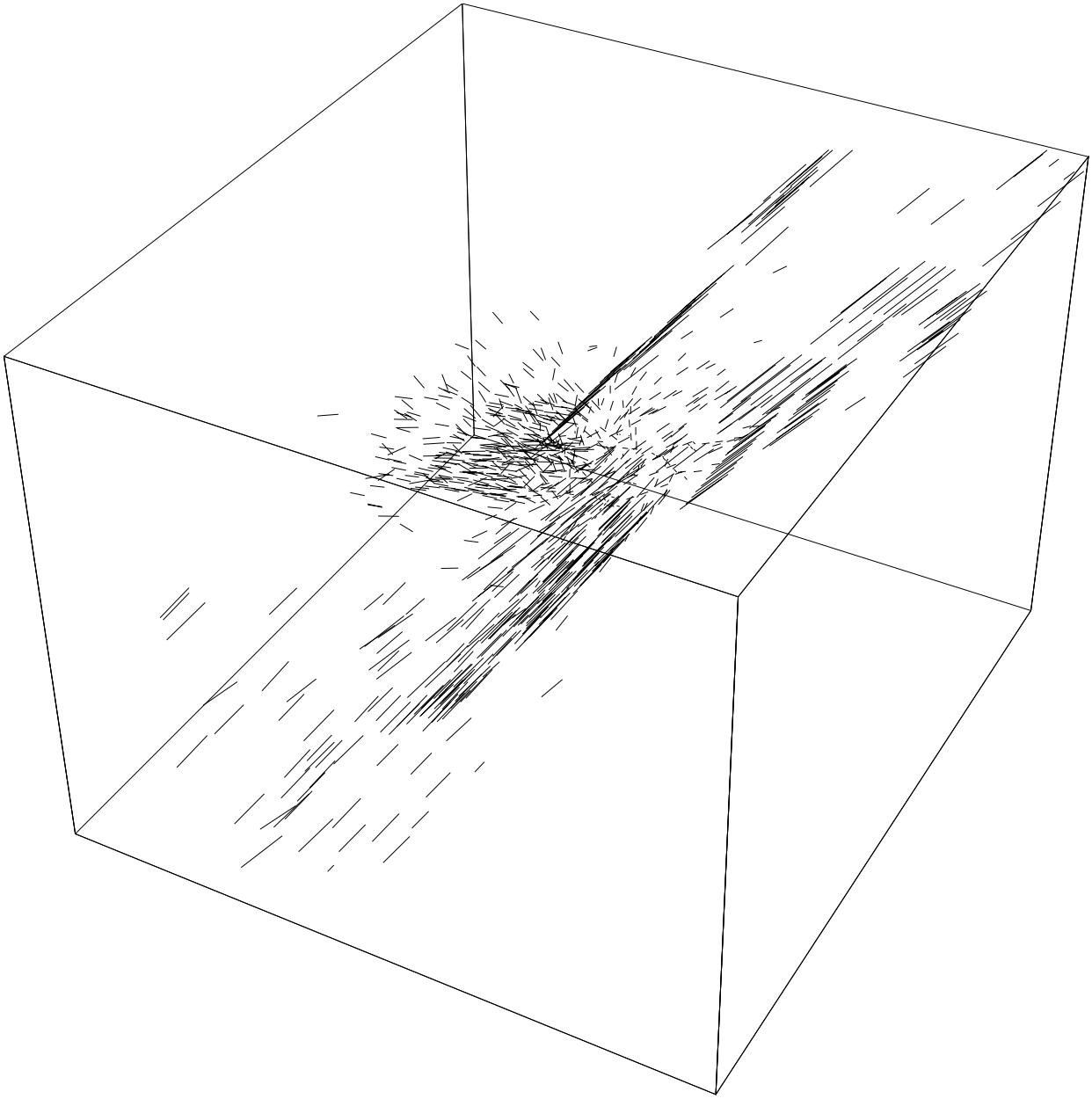
```
PlotChannel[TransformedSurfaces, 3]
```



Velocity vectors at points on transformed surfaces.

In[111]:=

```
FieldPlot =
 ListPlotVectorField3D[Flatten[Table[List[TransformedSurfaces[[t]][[i]][[j]],
     VelocityField[[1]][[i]][[j]]], {i, 1, Dimensions[TransformedSurfaces][[2]]},
    {j, 1, Dimensions[TransformedSurfaces][[3]]},
    {t, 1,  Dimensions[TransformedSurfaces][[1]]}], 2]]
```



Out[111]=

- Graphics3D -

■ Output Surfaces

melEncoding

Requires a 2D List of 3D vectors and integer number for labeling purposes. Begin formatting array of vectors for export by converting to a string and replacing ','s with a space. Initialize counters. 'i' for CV's in u direction. 'j' for CV's in v direction. 'd' gives size of image. Then move through lists and replace characters with mel code to select the surface and move the CV according to the SurfaceArray vectors. Finally add mel code header that creates the nurbs suface. Return String of mel code that creates nurbsPlane and moves CV's to locations in 2D list.

In[130]:=

```
melEncoding[surfaceList_List, imgNum_Integer] :=
 Module[{i, j, d, SurfaceArray, SurfaceArrayReplaced, SurfaceArrayFinal, melHeader},
  i = 0;
  j = 0;
  d = Dimensions[surfaceList];
  SurfaceArray = ExportString[surfaceList, "Table"];
  SurfaceArrayReplaced = StringReplace[SurfaceArray, {", " -> " "}];
  SurfaceArrayFinal = SurfaceArrayReplaced;
  For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
   replacePositions = StringPosition[SurfaceArrayFinal, "{"];
   SurfaceArrayFinal = StringReplacePart[SurfaceArrayFinal, "select $surface" <>
      ExportString[imgNum, "Text"] <> "; move ", replacePositions[[1]]];
  ];
  For[k = 1, k ≤ (d[[1]] * d[[2]]), k++,
   replacePositions = StringPosition[SurfaceArrayFinal, "}"];
   SurfaceArrayFinal =
    StringReplacePart[SurfaceArrayFinal, " .cv[" <> ExportString[i, "Text"] <>
      "][" <> ExportString[j, "Text"] <> "]; ", replacePositions[[1]]];
   If[i < d[[1]] - 1, i++, i = 0; j++;];
  ];
  melSurfaceHeader = "\n$surface" <> ExportString[imgNum, "Text"] <>
    "= `nurbsPlane -ch on -o on -po 0 -ax 0 1 0 -u " <>
    ExportString[(d[[1]] - 3), "Text"] <> " -v " <>
    ExportString[(d[[2]] - 3), "Text"] <> " -w 150 -lr 1`;\n";
  melSurfaceHeader <> SurfaceArrayFinal
 ]
```

melSurfaceCompile

Takes the list of surface coordinates and produces one surface at each time step.

In[131]:=

```
melSurfaceCompile[TransformedSurfaces_List] := Module[{melCode},
  melCode = "";
  For[n = 1, n ≤ Dimensions[TransformedSurfaces][[1]], n++,
    melCode = melCode <> melEncoding[N[TransformedSurfaces[[n]]], n];
   ]
  melCode
 ]
```

In[132]:=
```
melCode = melSurfaceCompile[TransformedSurfaces];
```

Export strings into .mel file.

In[133]:=
```
Export[projectPath <> sys <> iteration <> ".mel", melCode, "Text"]
```

Out[133]=
```
D:\My Documents\Columbia\Algorithmic Morphologies\Cellular03.mel
```

Export image of velocity vector field.

In[116]:=
```
Export[projectPath <> sys <> iteration <> "-01.jpg",
 FieldPlot, "JPEG", ImageSize → {800, 800}]
```

Out[116]=
```
D:\My Documents\Columbia\Algorithmic Morphologies\Cellular03-01.jpg
```