GPU-Accelerated Point Cloud Interpolation

Bo Zhou

Introduction

In natural science, there are many problems which are all could be merged as solving linear system, such as building engineering simulation, heat energy diffusion. Point cloud interpolation should be a very useful tool for solving PCA(Principal Component Analysis) problems, such as volume data interpolation, picture color reconstruction. Origin method, includes linear interpolation. We extend the method which was published in [1], use GPGPU to accelerate the basic linear algebra operation.

Solution

Assume there is a triangle T R_1, R_2, R_3, with a scalar value $q(x_i, y_i), i=1,2,3$ on each vertex. And we want to get interpolated value q_{ij} on a point x_{ij}, y_{ij} in the convex hull. This forms a linear interpolation function,

$$q_{in}(x,y)=q_1\phi_1(x,y)+q_2\phi_2(x,y)+q_3\phi_3(x,y) \quad .$$

The $\phi_i(x,y)=A_i/A$ is the area ratio, A_1 is the area of triangle $X R_2 R_3$. It's very clear that there are also 3 condition,

$$\begin{aligned}\phi_1(x,y)+\phi_2(x,y)+\phi_3(x,y) &= 1\\ x_1\phi_1(x,y)+x_2\phi_2(x,y)+x_3\phi_3(x,y) &= x\\ y_1\phi_1(x,y)+y_2\phi_2(x,y)+y_3\phi_3(x,y) &= y\end{aligned} \quad .$$

We re-write it as matrix form below,

$$\begin{pmatrix}1 & 1 & 1\\ x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{pmatrix}\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}=\begin{pmatrix}1\\ x\\ y\end{pmatrix} \quad .$$

The quadratic error basis function is

$$f(x,y)=a\phi_1\phi_2+b\phi_2\phi_3+c\phi_3\phi_1 \quad .$$

Let $S_j \in V, j=1,\cdots,m$ be the next m donor mesh points that are closest to X. Let $\phi_i(j)$ represent the value of ϕ_i at the data point S_j. Define a matrix

$$B=\begin{pmatrix}\phi_1(1)\phi_2(1) & \phi_2(1)\phi_3(1) & \phi_3(1)\phi_1(1)\\ \phi_1(2)\phi_2(2) & \phi_2(2)\phi_3(2) & \phi_3(2)\phi_1(2)\\ \vdots & \vdots & \vdots\\ \phi_1(m)\phi_2(m) & \phi_2(m)\phi_3(m) & \phi_3(m)\phi_1(m)\end{pmatrix} \quad .$$

To get the coefficients we invert a 3×3 system,

$$B^T Ba = B^T w$$

where $a=(a,b,c)^T$ and

$$w=\begin{pmatrix} q(1)-q_{lin}(1) \\ q(2)-q_{lin}(2) \\ \vdots \\ q(m)-q_{lin}(m) \end{pmatrix} .$$

It's very easy to get solutions in 3D case,

$$q(x,y,z)=q_{lin}(x,y,z)+f(x,y,z)$$

and

$$q_{lin}(x,y,z)=\sum_{i=1}^{4} q_i \phi_i(x,y,z) \ ,$$

inverse this matrix would get the $\phi_i, i=1,\cdots,4$,

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix}\begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}=\begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix} .$$

The quadratic error is

$$f(x,y,z)=a\phi_1\phi_2+b\phi_1\phi_3+c\phi_1\phi_4+d\phi_2\phi_3+e\phi_2\phi_4+f\phi_3\phi_4 \ .$$

The X is in a terahedron T , not planar triangle. Similar the (7) we can get 3D version of this linear system,

$$C^T Ca=C^T w$$

$$C=\begin{pmatrix} \phi_1(1)\phi_2(1) & \phi_2(1)\phi_3(1) & \phi_3(1)\phi_1(1) & \phi_2(1)\phi_3(1) & \phi_2(1)\phi_4(1) & \phi_3(1)\phi_4(1) \\ \phi_1(2)\phi_2(2) & \phi_2(2)\phi_3(2) & \phi_3(2)\phi_1(2) & \phi_2(1)\phi_3(1) & \phi_2(1)\phi_4(1) & \phi_3(1)\phi_4(1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_1(m)\phi_2(m) & \phi_2(m)\phi_3(m) & \phi_3(m)\phi_1(m) & \phi_2(1)\phi_3(1) & \phi_2(1)\phi_4(1) & \phi_3(1)\phi_4(1) \end{pmatrix}$$

after inverse the $C^T C$ we can easily obain the a . Third order and fourth order requires inverse 16×16 and 32×32 linear system.

Half Implementation

Current NVIDIA CUDA CUBLAS library has complete linear algebra support from BLAS1 to BLAS3. It's high efficient to operate matrix and vector in parallel on GPU now, so nearly the all work could be mapped onto GPU. There are maturity linear algebra operator LU released by some researchers from U.C. Berkeley and UNC. Use K-nearest to get some points near the interested position, then apply the above linear system to get the value.

Conclusion

Maybe you will think of that it's not valuable, simple linear interpolation is enough for grid data, to obtain the gradient. To achieve higher accuracy in engineering analysis, it's necessary to get interpolated point data not only in XYZ dimensions but any position. This idea is inspired by a paper about MAE and processing Houdini's fluid grid.