B 样条插值和拟合球

一、作业项目

- Approximate a sphere by using B-spline surface fitting and interpolation, and discuss the approximation accuracy.
- 使用 B 样条插值和拟合逼近一个球体，并讨论拟合精度。

二、编程环境

- 操作系统：Ubuntu 20.04.1 LTS (x64)
- 处理器：Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz
- 内存：8.00GB
- IDE：Clion 2020.2
- 编程语言：C++
- 第三方库：glut，Eigen。其中，使用 glut 显示球面，使用 Eigen 进行矩阵运算。

三、使用说明

请确保您的电脑上已经安装 OpenGl glut 库和 Eigen 库。安装方法见安装 OpenGL，安装 Eigen。

运行之后可以使用“w”,“a”,“s” ,“d”键控制球体旋转，按空格键切换显示模式，分为“显示所有”，“只显示曲面”，“只显示数据点”和“只显示控制点”四种模式。

1. 方法 1

直接使用 Clion 打开代码文件夹“BSpline”运行。

2. 方法 2

在命令行 release 文件夹下运行”./BSplien”程序 。

可选参数如下:

-m 拟合/插值中的球面纵向(经线方向)拟合/插值数据点的个数，正整数。

-n 拟合/插值中的球面纵横向(纬线方向)拟合/插值数据点的个数，正整数。

-M 逼近方式(拟合还是插值)，a 拟合， i 插值。

-g 绘制显示时使用的网格大小，在显示被逼近的球时本程序使用 g*g 大小的网格显示结果，数值越大显示效果越好(该参数影响显示效果不影响拟合/插值结果)，正整数。

运行示例:

```
“./BSpline -m 10 -n 10 -M i -g 20 ”
```

表示使用 B 样条插值球面，插值点数据的数量为(10+1) * (10+1)，显示时在绘制平面上选取(20+1) * (20+1)个点组成的网格表示。

四、算法描述

1. 拟合/插值数据点的生成

为了逼近一个球体，需要先生成球上需要拟合/插值的数据点。本程序根据输入的参数 m，n 分别在纵向（经线方向）上生成 m+1 个数据点，在横向（纬线方向）生成 n+1 个数据点。

数据点的坐标为：

$$0.5 * \left((1.0-\sin(\alpha)^2) * \sin(\theta), \quad \sin(\alpha), \quad (1.0-\sin(\alpha)^2) * \cos(\theta)\right)$$

其中α为数据点向量与$Z-X$平面的夹角，θ为数据点向量与$Y-Z$平面的夹角。

2. 拟合算法描述

对于拟合算法，需要先确定拟合数据点在 u(纵向)，v(横向)方向的个数 m+1 和 n+1，还需要确定 u 和 v 方向拟合控制点的个数 e+1、f+1 和拟合的次数 p、q。需要满足 m>e>=q>=1，n>f>=q>=1。本程序规定 e=m-1，f=n-1，p=e，q=f。

a) 参数向量 s，t 的生成

每一个拟合数据点在 u，v 方向都需要与一个参数对应。以 u 方向为例，先使用 **The Chord Length Method** 生成每一列拟合数据在该列中对应的参数，再用同一行所有拟合数据点参数的均值作为该行所有的点在 u 方向的参数。v 方向参数的确定方法与此相同。

b) 节点向量 knot_u，knot_v 的生成

程序中使用准均匀节点向量。knot_u 两端点节点重复度为 p+1，中间节点均匀分布。knot_v 两端节点重复度为 q+1，中间节点均匀分布。

c) 控制节点的生成

为了尽可能的逼近拟合数据点，需要让曲面数据点与对应的拟合数据点距离尽可能小。

对于 u 方向目标曲线方程为

$$\mathbf{C}(u)=\sum_{i=0}^{h}N_{i,p}(u)\mathbf{P}_i$$

我们令曲线经过第一个点和最后一个点，因此目标曲线方程为

$$\mathbf{C}(u)=N_{0,p}(u)\mathbf{P}_0+\sum_{i=1}^{h-1}N_{i,p}(u)\mathbf{P}_i+N_{h,p}(u)\mathbf{P}_h$$

原拟合数据点与目标曲线上对应数据点的距离函数为

$$f(\mathbf{P}_1,\dots,\mathbf{P}_{h-1})=\sum_{k=1}^{h-1}\left|D_k-C(t_k)\right|^2$$

(注意:方程右侧求和符号写错了，此处应该是 n 不是 h，下同)

将方程右侧展开可以得到

$$\begin{aligned}\mathbf{D}_k-C(t_k)&=\mathbf{D}_k-\left[N_{0,p}(u)\mathbf{P}_0+\sum_{i=1}^{h-1}N_{i,p}(u)\mathbf{P}_i+N_{h,p}(u)\mathbf{P}_h\right]\\&=\left(\mathbf{D}_k-N_{0,p}(u)\mathbf{P}_0-N_{h,p}(u)\mathbf{P}_h\right)-\sum_{i=1}^{h-1}N_{i,p}(u)\mathbf{P}_i\end{aligned}$$

令$\mathrm{Q_k}$为

$$\mathbf{Q}_k=\left(\mathbf{D}_k-N_{0,p}(u)\mathbf{P}_0-N_{h,p}(u)\mathbf{P}_h\right)$$

那么

$$f(\mathbf{P}_1,\dots,\mathbf{P}_{h-1})=\sum_{k=1}^{h-1}\left|\mathbf{Q}_k-\sum\nolimits_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right|^2$$

同时由于

$$
\begin{aligned}
&\left|\mathbf{Q}_k-\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right|^2\\
&=\left(\mathbf{Q}_k-\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)\left(\mathbf{Q}_k-\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)\\
&=\mathbf{Q}_k\mathbf{Q}_k-2\mathbf{Q}_k\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)+\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)
\end{aligned}
$$

因此$f(P_x)$可以写成

$$
f(\mathbf{P}_1,\dots,\mathbf{P}_{h-1})=\sum_{k=1}^{h-1}\left[\mathbf{Q}_k\mathbf{Q}_k-2\mathbf{Q}_k\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)+\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)\left(\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i\right)\right]
$$

对$f(P_1,P_2,\dots,P_{m-1})$求导并让导数等于0可得到

$$
\sum_{k=1}^{n-1}N_{g,p}(t_k)\sum_{i=1}^{h-1}N_{i,p}(t_k)\mathbf{P}_i=\sum_{k=1}^{n-1}N_{g,p}(t_k)\mathbf{Q}_k
$$

将方程写成矩阵形式可以得到

$$
\left(\mathbf{N}^T\mathbf{N}\right)\mathbf{P}=\mathbf{Q}
$$

其中

$$
\mathbf{P}=\begin{bmatrix}\mathbf{P}_1\\ \mathbf{P}_2\\ \vdots\\ \mathbf{P}_{h-1}\end{bmatrix}\qquad
\mathbf{Q}=\begin{bmatrix}\sum_{k=1}^{n-1}N_{1,p}(t_k)\mathbf{Q}_k\\ \sum_{k=1}^{n-1}N_{2,p}(t_k)\mathbf{Q}_k\\ \vdots\\ \sum_{k=1}^{n-1}N_{h-1,p}(t_k)\mathbf{Q}_k\end{bmatrix}\qquad
\mathbf{N}=\begin{bmatrix}N_{1,p}(t_1) & N_{2,p}(t_1) & \cdots & N_{h-1,p}(t_1)\\ N_{1,p}(t_2) & N_{2,p}(t_2) & \cdots & N_{h-1,p}(t_2)\\ \vdots & \vdots & \ddots & \vdots\\ N_{1,p}(t_{n-1}) & N_{2,p}(t_{n-1}) & \cdots & N_{h-1,p}(t_{n-1})\end{bmatrix}
$$

矩阵方程中N和Q已知，因此可以通过解矩阵方程得控制节点矩阵P。

在每一列使用以上算法计算一次，可以得到一个控制节点矩阵P′。再将P′视作新的拟合数据点，在每一行上重复该算法，即可求得曲面样条拟合的最终控制节点矩阵P。

3. 插值算法描述

与拟合算法不同的是，插值算法规定了控制节点数与插值点数相等，因此对于插值算法可以直接通过一次矩阵计算得到控制点，而不需要像拟合算法一样对每个方向分别计算。具体描述如下：

a) 参数向量 s，t 的生成

与拟合算法时生成 s，t 参数的方法一样。

b) 节点向量 knot_u，knot_v 的生成

为了让计算矩阵非奇异，在生成 knot 向量时使用 de Boor 提出的 average parameters 方法，计算公式如下：

$$
u_0=u_1=\cdots=u_p=0
$$

$$
u_{j+p}=\frac{1}{p}\sum_{i=j}^{j+p-1}t_i \qquad \text{for } j=1,2,\cdots,n-p
$$

$$
u_{m-p}=u_{m-p+1}=\cdots=u_m=1
$$

c) 控制节点的生成

假设结果曲面的方程为

$$\mathbf{S}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{P}_{ij}N_{i,p}(u)N_{j,q}(v)$$

因为曲面$S(u,v)$经过插值数据点，因此有

$$\begin{aligned}\mathbf{D}_{cd}=\mathbf{S}(s_c,t_d)&=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{P}_{ij}N_{i,p}(s_c)N_{j,q}(t_d)\\&=\sum_{i=0}^{m}N_{i,p}(s_c)\left(\sum_{j=0}^{n}\mathbf{P}_{ij}N_{j,q}(t_d)\right)\end{aligned}$$

令Q_{id}为

$$\mathbf{Q}_{id}=\sum_{j=0}^{n}\mathbf{P}_{ij}N_{j,q}(t_d)$$

那么有

$$\mathbf{D}_{cd}=\sum_{i=0}^{m}N_{i,p}(s_c)\mathbf{Q}_{id}$$

我们可以将其写成矩阵形式得到

$$\mathrm{D}=\mathrm{N}_{\mathrm{i,p}}\mathrm{P}\mathrm{N}_{\mathrm{j,q}}$$

其中D，$N_{i,p}$和$N_{j,q}$都已知，因此可以通过矩阵运算解得控制点矩阵P。

4. 结果曲面数据的计算

在得到控制点P和节点向量$knot$之后，可以根据 B 样条曲面方程计算曲面上点的位置。曲面方程如下。

$$\mathbf{S}(u,v)=\sum_{i=0}^{e}\sum_{j=0}^{f}\mathbf{P}_{ij}N_{i,p}(u)N_{j,q}(v)$$

为了降低计算成本，本文使用在曲面上选取$(g+1)*(g+1)$个在$([0,1],[0,1])$范围内的均匀数据点作为显示的点。

5. 曲面闭合的处理

为了得到一个闭合的 B 样条曲面，尽可能地逼近球体，本程序重复使用纬线方向的第一个数据点，在经线方向上，通过在极点处添加两个点$(0,0.5,0)$和$(0,-0.5,0)$，使曲面在经线方向上闭合。

五、数据结构说明

1. Point 结构体：

```
struct Point {
    float x;
    float y;
    float z;
    Point() {
        x = y = z = 0.0f;
    }
    Point(float x, float y, float z) : x(x), y(y), z(z) {}
};
```

表示空间中的一个点，包含x,y,z三个 float 类型变量，表示坐标。

2. BSpline 类：

BSpline 类 B 样条插值/拟合曲面类，根据用户输入的参数计算样条函数所需要的参数、节点向量和控制节点。再根据曲面方程计算插值/拟合曲面上的各点。

此处只给出计算控制节点和计算曲面上点的函数部分代码：

3. calculateControlPointsInter()

计算插值方法时需要用到的控制点：

```
void calculateControlPointsInter() {

    // init s_, t_
    initST();
    // init u_, v_
    initUVInter();
    // init Nsi, Njt
    initBaseFunction();

    vector<MatrixXf> data_mat(3, MatrixXf(m_ + 1, n_ + 1));
    // init data_x_y_z;
    for (int i = 0; i < m_ + 1; i++) {...
    }
    for (int i = 0; i < 3; i++) {
        MatrixXf Q = base_function_s_i_.colPivHouseholderQr().solve(data_mat[i]);
        MatrixXf P = base_function_j_t_.transpose().colPivHouseholderQr().solve(Q.transpose());
        data_mat[i] = P.transpose();
    }

    vector<vector<Point>> temp_vec_con(m_ + 1, vector<Point>(n_ + 1));
    control_points_.assign(temp_vec_con.begin(), temp_vec_con.end());

    for (int i = 0; i < m_ + 1; i++) {
        for (int j = 0; j < n_ + 1; j++) {
            control_points_[i][j].x = data_mat[0](i, j);
            control_points_[i][j].y = data_mat[1](i, j);
            control_points_[i][j].z = data_mat[2](i, j);
        }
    }

    calculateShowPoints();
    fillEmpty();

    calError();
};
```

4. calculateControlPointsAppro()

计算拟合球面时需要用到的控制点，下面给出计算一列数据点对应控制点的代码：

```
for (int col = 0; col < n_ + 1; col++) {
    float P0 = D(0, col);
    float Pe = D(m_, col);

    VectorXf P(e_ - 1);
    VectorXf QK(m_ - 1);
    VectorXf Q(e_ - 1);

    MatrixXf N(m_ - 1, e_ - 1);
    // init N
    for (int i = 0; i < m_ - 1; i++) {
        for (int j = 0; j < e_ - 1; j++) {
            N(i, j) = getN(j + 1, p_, s_[i + 1], u_);
        }
    }
    // init QK
    for (int i = 0; i < m_ - 1; i++) {
        int k = i + 1;
        QK(i) = D(k, col) - getN(0, p_, s_[k], u_) * P0 - getN(e_, p_, s_[k], u_) * Pe;
    }
    // inti Q
    Q = (QK.transpose() * N).transpose();
    // NT*N*P = Q
    MatrixXf NTN = N.transpose() * N;
    P = NTN.lu().solve(Q);
    temp_D(0, col) = P0;
    temp_D(e_, col) = Pe;

    for (int i = 0; i < e_ - 1; i++) {
        temp_D(i + 1, col) = P(i);
    }
}
```

5. getN(int, int, float, vector<float>)

用来计算$N_{i,p}$和N_{jq}。

```
float getN(const int i, const int p, const float t, vector<float> &u) {
    if (p == 0) {
        return (t >= u[i] && t < u[i + 1]) ? 1.0f : 0.0f;
    }
    float left = u[i + p] - u[i];
    float right = u[i + p + 1] - u[i + 1];

    if (globalFunction::floatEqual(left, 0.0f)) {
        left = (t - u[i]) * getN(i, p - 1, t, u);
    } else {
        left = (t - u[i]) / left * getN(i, p - 1, t, u);
    }
    if (globalFunction::floatEqual(right, 0.0f)) {
        right = (u[i + p + 1] - t) * getN(i + 1, p - 1, t, u);
    } else {
        right = (u[i + p + 1] - t) / right * getN(i + 1, p - 1, t, u);
    }
    return left + right;
}
```

6. calculateShowPoints()

用来根据曲面方程计算曲面上的点的位置。

```
void calculateShowPoints() {

    show_points_.resize(num_points_on_line_ + 1);

    for (int i = 0; i <= num_points_on_line_; i++) {
        for (int j = 0; j <= num_points_on_line_; j++) {
            float s = ((float) i) / num_points_on_line_;
            float t = ((float) j) / num_points_on_line_;

            if (globalFunction::floatEqual(s, 1.0f)) {
                s = 1.0f - Epsilon * 2;
            }
            if (globalFunction::floatEqual(t, 1.0f)) {
                t = 1.0f - Epsilon * 2;
            }

            // s,t
            Point p(0.0f, 0.0f, 0.0f);
            for (int x = 0; x < control_points_.size(); x++) {
                float n_i_p = getN(x, p_, s, u_);
                for (int y = 0; y < control_points_[x].size(); y++) {
                    float n_j_q = getN(y, q_, t, v_);
                    p.x += n_i_p * n_j_q * control_points_[x][y].x;
                    p.y += n_i_p * n_j_q * control_points_[x][y].y;
                    p.z += n_i_p * n_j_q * control_points_[x][y].z;
                }
            }
            show_points_[i].push_back(p);
        }
    }
}
```

六、运行结果与分析

1. B 样条拟合球面结果分析

使用不同的拟合数据点数量对球面进行拟合，得到的最终结果曲面与拟合数据点的误差如下表显示：

拟合数据点个数			曲线次数		拟合误差
u方向	v方向	总数	p	q	
5	5	25	3	3	6.21%
6	6	36	3	3	1.65%
7	7	49	3	3	0.66%
8	8	64	3	3	0.20%
9	9	81	3	3	0.07%
10	10	100	3	3	0.02%
11	11	121	3	3	0.00%

表 1. 不同拟合数据点个数下的数据点拟合误差

从表格中可以看出，在使用 36 个数据点进行拟合球时，对于每个被拟合点，平均误差已经达到 1.65%。但是可以观察到拟合的球与真实的球仍存在差距。

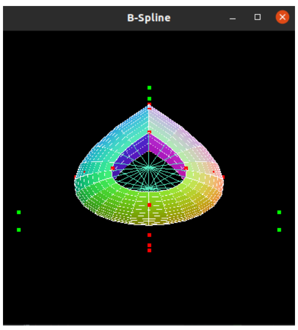
在拟合曲面上均匀采样 400 个点，设 u，v 方向曲线次数为 3，计算曲面上采样点与真实值的误差，详细数据见表。

拟合数据点个数			采样数据点个数	误差
u方向	v方向	总数		
5	5	25	400	7.94%
6	6	36	400	3.41%
7	7	49	400	1.30%
8	8	64	400	0.39%
9	9	81	400	0.13%
10	10	100	400	0.06%
11	11	121	400	0.03%

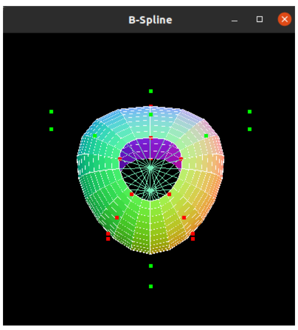
表 2. 在曲面上 400 个数据点与真实值的平均误差

可以看到，随着拟合点数目的增加，拟合曲面上采样数据点误差不断减小，在使用 36 个点拟合球面时，从球面上均匀采样 400 个数据点，平均误差为 3.41%，当使用 64 个拟合点拟合曲面时，误差会将为 0.39%，肉眼基本观察不出拟合曲面和真实球面的差别。

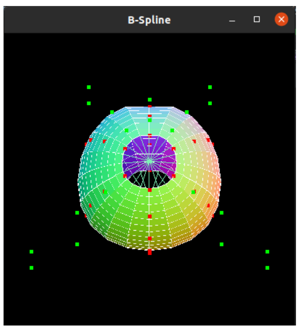
下面是不同拟合点数时的绘制结果：



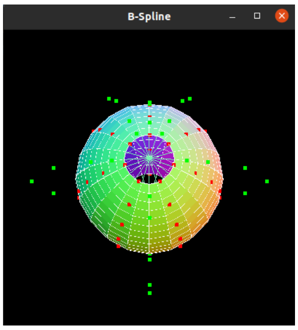
a. 5*5 个拟合数据点



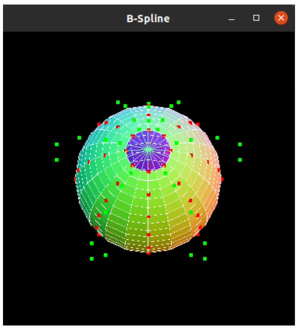
b. 6*6 个拟合数据点



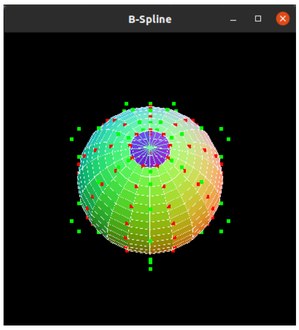
c. 7*7 个拟合数据点



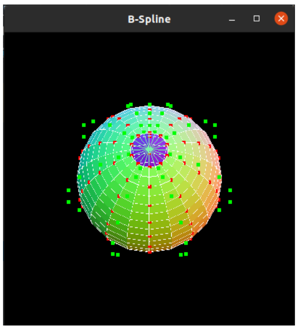
d. 8*8 个拟合数据点



e. 9*9 个拟合数据点



f. 10*10 个拟合数据点



g. 11*11 个拟合数据点

2. B 样条插值球面结果

由于插值曲面会经过所有的插值点，因此各插值点处的误差都为 0。

类似的，使用 3 次插值曲面，不同的插值数据点个数时曲面上 400 采样点平均误差结果如下：

插值数据点个数			采样数据点个数	误差
u方向	v方向	总数		
5	5	25	400	3.52%
6	6	36	400	1.13%
7	7	49	400	0.50%
8	8	64	400	0.23%
9	9	81	400	0.13%
10	10	100	400	0.07%
11	11	121	400	0.03%

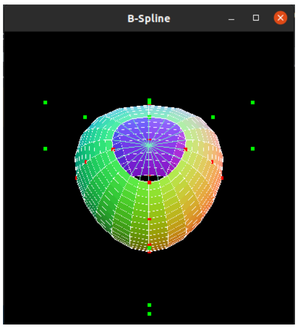
表 3. 在曲面上 400 个数据点与真实值的平均误差

与拟合时相同，随着插值数据点个数的增加误差逐渐变小，，当使用 49 个插值点生成曲面时，误差已经降到 0.50%，插值曲面跟真实球面的差别已经很小。

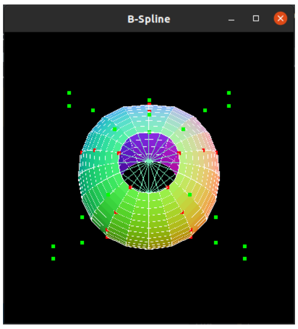
比较拟合和插值两种方法时可以发现，在使用相同数量的数据点、曲面次数也相同时，插值曲面的误差普遍比拟合曲面的误差小。这是因为插值可视为拟合曲面方程的一个解，插值曲面使用的控制点个数比拟合时的控制点个数多，因此对曲面的控制能力更好，误差更小。

因此在用 B 样条曲面描述球面时，选用插值方法效果会更好。

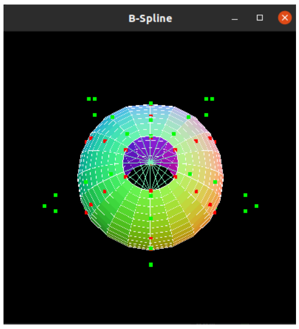
下面是不同插值点数时的绘制结果:



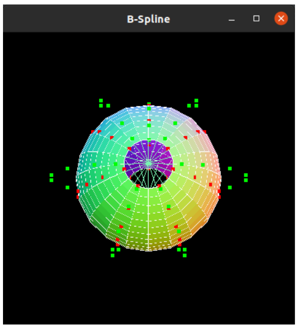
a. 5*5 个插值数据点



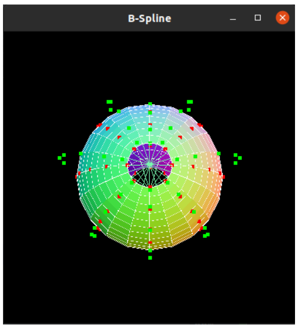
b. 6*6 个插值数据点



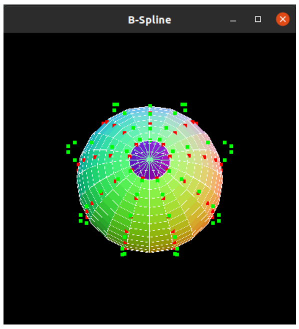
c. 7*7 个插值数据点



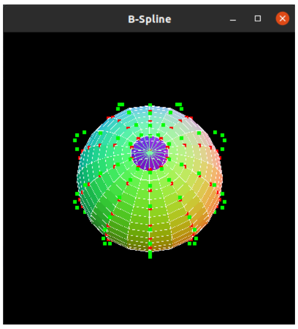
d. 8*8 个插值数据点



e. 9*9 个插值数据点

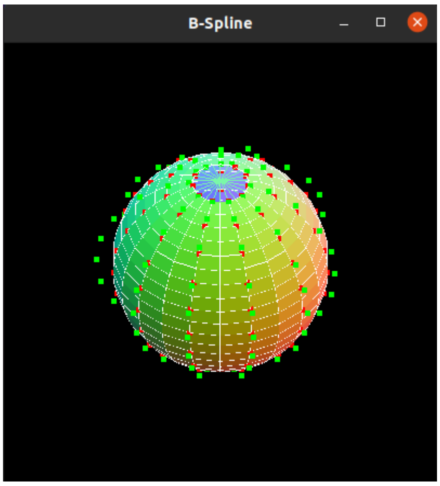


f. 10*10 个插值数据点

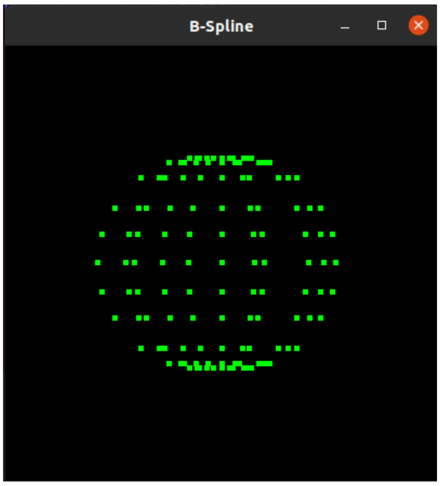


g. 11*11 个插值数据点

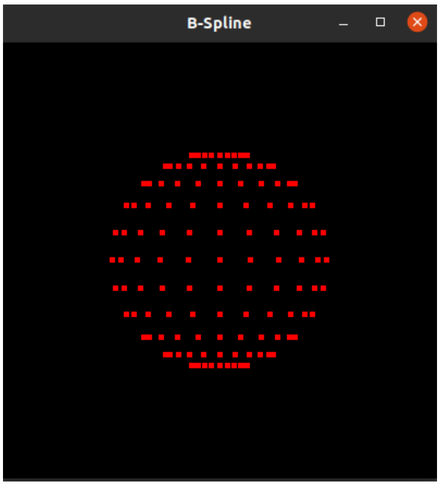
3. 不同显示模式下显示结果



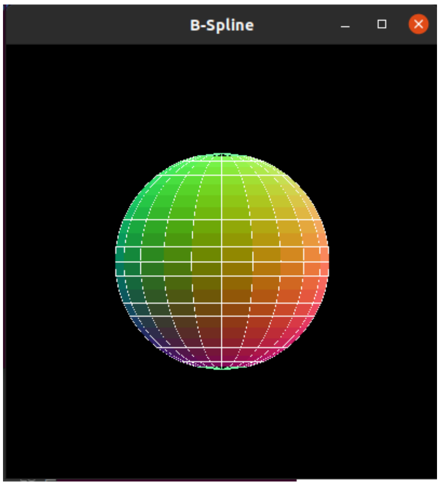
a.显示所有数据



b.只显示控制节点



c.显示插值数据点



d.只显示曲面

七、参考文献

[1] Zhang, Hongxin, and Jieqing Feng. "B-spline interpolation and approximation." *Digital Geometry Processing course lecture notes, Zhejiang University* (2006).