

计算机图形学 课程项目

王徐笑风* 凌泽辉†

2020 年 11 月 5 日

*学号:18120193 E-mail:2208740924@qq.com

†学号:18120193 E-mail:785896610@qq.com

目录

1	三维图形的变换与表示	4
1.1	向量与点	4
1.2	变换	4
1.2.1	平移	4
1.2.2	旋转	5
1.2.3	缩放	6
1.2.4	复合变换	7
1.3	三维的齐次坐标	7
1.3.1	<i>sunMatLib.h</i> 矩阵运算库	8
1.4	三角面	11
1.5	网格	12
1.5.1	.obj 文件的加载	12
2	投影	14
2.1	正交投影	14
2.2	透视投影	14
3	相机变换	14
3.1	世界坐标系	14
3.2	视口坐标系	14
3.3	坐标系变换	14
4	绘制/光栅化	14
4.1	三角填充	14
4.2	绘制顺序问题	14
4.3	画家算法	14
4.3.1	画家算法的问题	14
4.3.2	改进方向	14
5	三维裁剪	15
5.1	性能问题与原因	15

目 录	3
5.2 三角面裁剪	15
5.2.1 近平面裁剪	15
5.2.2 视口裁剪	15
6 基础光照	15
6.1 全局光照	15
6.2 方向光	15

摘要

查找资料，学习了解三维网格模型的相关知识。完成一个三维网格模型的显示系统。

数据输入：通过文件读取模型数据

数据存储：设计程序内用于存储模型数据的数据结构

数据输出：在窗口界面进行模型显示

编程实现三维到二维的投影变换计算

编程实现通过键盘或鼠标驱动模型的平移、缩放及旋转变换

可以使用开发工具中提供光照函数，若自己编程实现光照计算，则可获得额外加分

1 三维图形的变换与表示

1.1 向量与点

在三维空间中我们常用一个三维向量表示一个点，虽然向量本身只表达长度和方向，他是无关坐标系的，而点显然是与选取的坐标系是相关的。因此在这里将点理解为在一个给定的坐标系下，原点按某一向量移动后的位置，它写作式 (1)：

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (1)$$

1.2 变换

对于三维空间中的点，常用到的仿射变换和二维中的类似：平移、旋转和缩放。

1.2.1 平移

平移变换是将一个点按一个方向，移动一段距离。考虑到上面我们的点的定义即为在给点的坐标系下，原点按一个向量移动的距离。那么显然平移一个点即将原点平移两次，即点所代表的“向量”和平移向量的共同作用。

对于点 \mathbf{P} ，将其平移 \mathbf{V} ：

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_x \\ \mathbf{P}_y \\ \mathbf{P}_z \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_x \\ \mathbf{V}_y \\ \mathbf{V}_z \end{bmatrix}$$

则有平移后的点 \mathbf{N} ：

$$\mathbf{N} = \begin{bmatrix} \mathbf{P}_x \\ \mathbf{P}_y \\ \mathbf{P}_z \end{bmatrix} + \begin{bmatrix} \mathbf{V}_x \\ \mathbf{V}_y \\ \mathbf{V}_z \end{bmatrix} = \begin{bmatrix} \mathbf{P}_x + \mathbf{V}_x \\ \mathbf{P}_y + \mathbf{V}_y \\ \mathbf{P}_z + \mathbf{V}_z \end{bmatrix}$$

1.2.2 旋转

旋转指的是：点以三维空间中的某点为旋转中心，进行旋转，这里简略的认为旋转中心为坐标系原点，即此时的旋转变换是一个特殊的线性变换。在三维坐标系中对点做按原点的旋转，即是对一个向量进行旋转。只需要求取原基向量 \hat{i} 、 \hat{j} 、 \hat{k} ，在旋转后的 \hat{i}' 、 \hat{j}' 、 \hat{k}' ，可以得到旋转矩阵：

$$\text{RotateMatrix} = \begin{bmatrix} \hat{i}'_x & \hat{j}'_x & \hat{k}'_x \\ \hat{i}'_y & \hat{j}'_y & \hat{k}'_y \\ \hat{i}'_z & \hat{j}'_z & \hat{k}'_z \end{bmatrix}$$

则有旋转后的点 \mathbf{N} ：

$$\mathbf{N} = \begin{bmatrix} \hat{i}'_x & \hat{j}'_x & \hat{k}'_x \\ \hat{i}'_y & \hat{j}'_y & \hat{k}'_y \\ \hat{i}'_z & \hat{j}'_z & \hat{k}'_z \end{bmatrix} \times \begin{bmatrix} \mathbf{P}_x \\ \mathbf{P}_y \\ \mathbf{P}_z \end{bmatrix}$$

特别的，绕 Y 轴旋转 θ 弧度的旋转矩阵，可以这么考虑：首先 Y 轴显然是不变的，在左手系下从 Y 轴逆方向向下看，Z 轴 X 轴正好组成一个平面直角坐标系。则 \hat{i} 顺时针旋转过 θ 弧度后的向量为：

$$\hat{i}' = \begin{bmatrix} \cos(\theta) \\ 0 \\ -\sin(\theta) \end{bmatrix}$$

同样的 \hat{k} 旋转过 θ 弧度后的向量为:

$$\hat{k}' = \begin{bmatrix} \sin(\theta) \\ 0 \\ \cos(\theta) \end{bmatrix}$$

综上, Y 轴旋转矩阵为:

$$\mathbf{RotateY}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2)$$

类似的我们也可以得到 X 轴旋转和 Z 轴旋转矩阵:

$$\mathbf{RotateX}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (3)$$

$$\mathbf{RotateZ}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

最后, 通过上述 (3)、(2)、(4) 的旋转矩阵的复合矩阵即可实现任意的旋转, 即:

$$\mathbf{Rotate}(\alpha, \beta, \gamma) = \mathbf{RotateZ}(\gamma) \times \mathbf{RotateY}(\beta) \times \mathbf{RotateX}(\alpha)$$

1.2.3 缩放

缩放的实现和旋转矩阵类似, 计算出新的 \hat{i}' 、 \hat{j}' 、 \hat{k}' 即可:

$$\mathbf{Scale}(\alpha, \beta, \gamma) = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix}$$

则缩放后的点为:

$$\mathbf{N} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \alpha x \\ \beta y \\ \gamma z \end{bmatrix}$$

1.2.4 复合变换

不难看出上述的变换中，旋转和缩放的变换可以简单地实现符合，将旋转矩阵和缩放矩阵进行矩阵乘法复合即可，同时这里两者是可交换的。

但平移变换就无法使用矩阵表达（矩阵变换后的点 \mathbf{N} 的一个分量可以表达为 $\mathbf{N}_x = a\mathbf{P}_x + b\mathbf{P}_y + c\mathbf{P}_z$ 而平移变换可表达为 $\mathbf{N}_x = \mathbf{P}_x + d$ 显然上述公式中 $a = 1, \mathbf{P}_y + c\mathbf{P}_z = d$ 显然无法使用一个静态的矩阵实现），因此在描述一个点的平移、旋转和缩放变换时，我们使用下述公式 (5)

$$\mathbf{N} = (\mathbf{Scale}(a, b, c) \times \mathbf{Rotate}(\alpha, \beta, \gamma)) \times \mathbf{P} + \mathbf{V} \quad (5)$$

但使用上述公式计算是不“简单”的，我们希望能有一种方法通过一次一种计算即可表达上述三种变换，同时又能提高运算的效率。

1.3 三维的齐次坐标

在三维中我们无法将平移、旋转和缩放变换由一个矩阵描述。不妨假设我们在四维中，并规定点 \mathbf{P} 在四维中的坐标为：

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ z \\ w = 1 \end{bmatrix}$$

即对于 $\forall \mathbf{P}$ 他们位于四维空间中分量 $w = 1$ 的一个三维“切片”空间中。而对于 $w \neq 1$ 的点，可以通过计算 $x' = \frac{x}{w}, y' = \frac{y}{w}, z' = \frac{z}{w}$ 来得到对应单位空间中的坐标。

有了三维齐次坐标后，我们可以改写公式 (5) 为：

$$\mathbf{N} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\text{Transform}} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\mathbf{P}} = \begin{bmatrix} ax + by + cz + d \\ ex + fy + gz + h \\ ix + jy + kz + l \\ 1 \end{bmatrix} \quad (6)$$

观察公式 (6)，其中 d, h, l 显然对应了平移向量：

$$\mathbf{V} = \begin{bmatrix} d \\ h \\ l \\ 1 \end{bmatrix}$$

它可以写成一个平移矩阵：

$$\mathbf{Translate} = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

而将平移矩阵和旋转缩放矩阵复合即可得到公式 (6) 中的变换矩阵：

$$\begin{aligned} \mathbf{Translate} \times \mathbf{Scale} \times \mathbf{Rotation} &= \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\mathbf{Translate}} \times \begin{bmatrix} a & b & c & 0 \\ e & f & g & 0 \\ i & j & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\mathbf{Scale} \times \mathbf{Rotation}} \\ &= \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\mathbf{Transform}} \end{aligned}$$

综上所述我们通过三维空间中的齐次坐标实现了 3 种变换的复合。当然 *Transform* 矩阵也能表达切变等仿射变换，由于在三维投影的过程中并不常用，这里就不详细叙述。

1.3.1 *sunMatLib.h* 矩阵运算库

在本项目中，除了窗口创建和像素绘制使用到了第三方软件库，所有的数据结构、矩阵矩阵运算、投影、光栅化、光照等都有我们小组自己实现。这里简单介绍一下整个项目中频繁使用到的矩阵运算类库，见源码：


```
1 class Vector3
2 {
3     friend class Matrix4;
4
5 public:
6     double _x, _y, _z, _w;
7
8 public:
9     Vector3();
10    Vector3(const double &x, const double &y, const
        double &z);
11    Vector3(const Vector3 &copy);
12
13    Vector3 &operator=(const Vector3 &copy);
14    Vector3 operator+(const Vector3 &b) const;
15    Vector3 operator-(const Vector3 &b) const;
16    Vector3 operator*(const double &a) const;
17    friend Vector3 operator*(const double &a, const
        Vector3 &b);
18    double operator*(const Vector3 &b) const;
19    Vector3 &operator*=(const double &a);
20    Vector3 &operator+=(const Vector3 &b);
21    friend Vector3 operator/(const Vector3 &b, const
        double &a);
22    friend Vector3 &operator/=(Vector3 &b, const
        double &a);
23    double length() const;
24    Vector3 normalize() const;
25    Vector3 &normalized();
26    Vector3 cross(const Vector3 &b) const;
27    Vector3 &crossed(const Vector3 &b);
```

```
28     Vector3 &set(const double &x, const double &y,
29                const double &z);
30
31     static Vector3 ONE();
32     static Vector3 ZERO();
33     static Vector3 UP();
34     static Vector3 FRONT();
35     static Vector3 RIGHT();
36 };
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
21     static Matrix4 ROTATE_Z(const double &angle);
22     static Matrix4 ROTATE(const double &x, const
        double &y, const double &z);
23     static Matrix4 SCALE(const double &x, const
        double &y, const double &z);
24     static Matrix4 TRANSLATE(const double &x, const
        double &y, const double &z);
25     static Matrix4 POINTAT(Vector3 pos, Vector3
        target, Vector3 up);
26 };
```

1.4 三角面

虽然三维形体有很多表示方法，但在三维的渲染引擎或是游戏引擎中，最常使用到的还是三角网格模型。

在这种网格模型中一个面一个三维空间中的三角面，它由 3 个顶点 $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ 即可表达。

保留法向信息，三角化的 *.obj* 文件就是这样一种三角网格模型，在本项目中也该文件格式作为标准。

特别的，在三维渲染中，单个三角面往往还有“朝向”这一属性。背对我们的三角面是不显示的，只有面对我们的才会渲染。这种规定在处理封闭模型和时候会更加有效：首先我们是无法看到封闭模型的内部的，自然无需渲染其三角面的背面；第二，背对我们的三角面不进行渲染也能降低对于性能的需求。因此三角面还存在一个法向量，它垂直于三角面所在的平面，并与三角面朝向同向。

但这不代表我们需要使用除 3 个三维点以外的数据保存法向信息。在 *.obj* 中一个三角面 *face* 由 3 个顶点 *vertex* 组成。这三个顶点按原三角面的逆时针排列，其法向满足右手螺旋定则，因此对于 1 个三角面的有序顶点 $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ ，该三角面的单位法向量 \mathbf{U} 为：

$$\mathbf{U} = \frac{(\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)}{|(\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)|}$$

在 *sunMatLib.h* 也有三角面类 *TriFace*，它实现了法向的自动计算，在通过顶点构造了三角面后，即可通过 *getNormal()* 方法获得单位法向量。具体实现可见源程序代码。

1.5 网格

在有了三角面的数据结构后，三角网格模型的最简单的实现就是使用变长数组保存每一个三角面。

在 *sun3D.h* 中的 *Mesh* 类就是使用了 *std::vector<TriFace>* 来保存三角网格模型。

1.5.1 .obj 文件的加载

.obj 文件的构成或是说格式，基本上——对应了上述的三角面、三角网格模型的数据结构。我们先来看一个 .obj 文件实例 (*untitled1.obj* 文件有省略)：

```
1 # Blender v2.83.4 OBJ File: ''
2 # www.blender.org
3 o Torus
4 v 1.250000 0.000000 0.000000
5 v 1.216506 0.125000 0.000000
6 v 1.125000 0.216506 0.000000
7 v 1.000000 0.250000 0.000000
8 .....
9 s off
10 f 13 2 1
11 f 2 15 3
12 f 15 4 3
13 f 16 5 4
14 .....
```

其中“#”开头的是注释，忽略即可。

“v”开头的一行定义了一个顶点，后跟三个数值分别表示该顶点 \mathbf{P} 的 x, y, z 分量

“f”开头的一行定义了一个三角面，后跟三个整数分别表示该三角面的 3 个顶点 $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ 所对应的顶点“v”的序号。例如：上述文件中的第 10 行“f 13 2 1”，其中“2 1”表明 \mathbf{P}_2 为第 5 行的顶点， \mathbf{P}_3 为第 4 行的顶点。

因此，读取 .obj 文件十分简单，设置一个 `std::vector<Vector3>` 来保存顶点，当读取到一行以“v”开头就构造一个 `Vector3` 顶点，并放入 `std::vector<Vector3>` 中，当读取到一行以“f”开头就从 `std::vector<Vector3>` 中依次取出相应下标的顶点，构造三角面，并放入需要的 `Mesh` 实例中的 `std::vector<TriFace>` 即可。

2 投影

想要实现三维模型的显示，最重要的是找到一种将三维模型“转换”/“映射”/“投影”到二维平面的方法。

2.1 正交投影

2.2 透视投影

3 相机变换

3.1 世界坐标系

3.2 视口坐标系

3.3 坐标系变换

4 绘制/光栅化

4.1 三角填充

4.2 绘制顺序问题

4.3 画家算法

4.3.1 画家算法的问题

4.3.2 改进方向

5 三维裁剪

5.1 性能问题与原因

5.2 三角面裁剪

5.2.1 近平面裁剪

5.2.2 视口裁剪

6 基础光照

6.1 全局光照

6.2 方向光