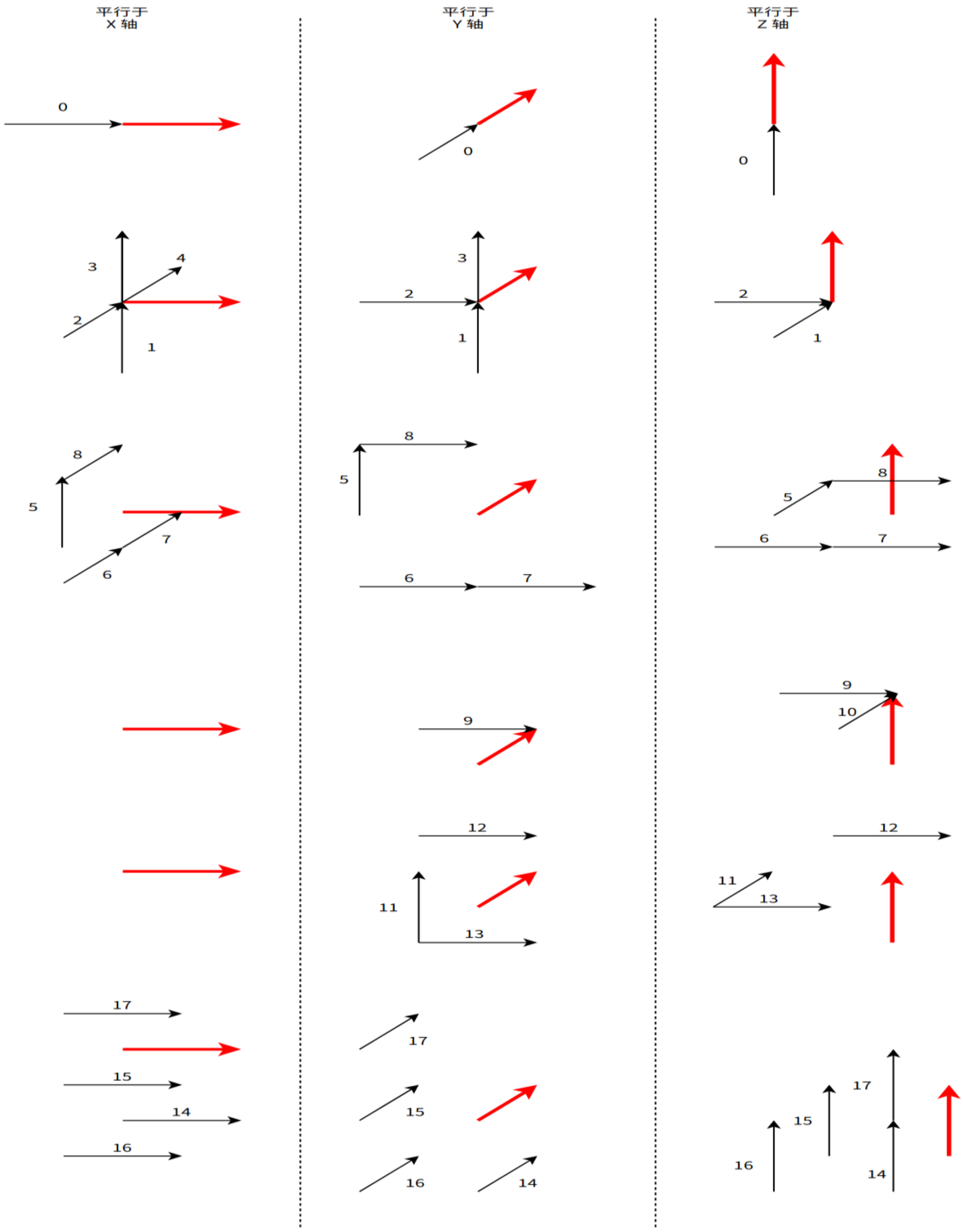
2.27——次要信息顺序优化

V20 次要上下文信息熵值统计

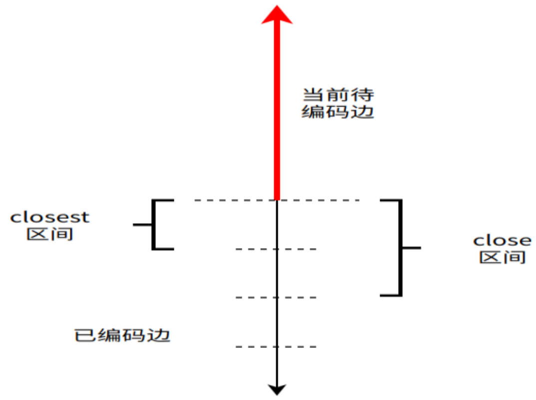
1. 18 条边



2. 三个不同方向用到的不同 9 条边

```
int mapping18to9[3][9] = { {0,1,2,3,4,15,14,5,7}, {0,1,2,3,9,15,14,7,12}, {0,1,2,9,10,15,14,7,12} };
```

3. 边的区间划分



优化方案的理论基础

1. 获取使用某一上下文时，边的占据情况信息
2. 计算每个序列，每个码率点，每个 slice 下，不同上下文值的熵值
3. 每个码率点取加权平均，得到平均熵值
4. 对四种类型点云数据，在 C2 条件下进行了测试
5. 熵值越小说明上下文越有效

编码顶点标识信息用到的次要信息

- $neighbEnd$：表示包含当前待编码边终止点的 4 个节点的占据情况
- $patternClose$：9 条边在 close 区间被占据的情况
- $direction$：当前待编码边的朝向（0=X，1=Y，2=Z）
- $pattern$：9 条边的被占据的情况
- $orderedPclosePar$： 不相邻的平行与垂直边

现有顺序

```
//int ctxMap2 = neighbEnd << 11;
//ctxMap2 |= (patternClose & (0b00000110)) << 9 - 1 ; // perp that do not depend on direction = to start
//ctxMap2 |= direction << 7;
//ctxMap2 |= (patternClose & (0b00011000))<< 5-3; // perp that  depend on direction = to start or to end
//ctxMap2 |= (patternClose & (0b00000001))<< 4;  // before
//int orderedPclosePar = (((pattern >> 5) & 3) << 2) + (!!(pattern & 128) << 1) + !!(pattern & 256);

//ctxMap2 |= orderedPclosePar;
```

neighbEnd>>patternClose&0b00000110(P_1)>>direction>>patternClose&0b00011000(P_2)>>patternClose&0b00000001(P_3)>>orderedPclosePar

改进方案一

按照熵值越小，上下文效果越好的理论基础。先通过测试得出各个上下文在各个序列以及不同码率点下的熵值，然后作图分析其应有的顺序。

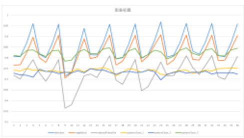
solid 类型序列测试结果

次要信息名称	direction	neighbEnd	orderedPclosePar	patternClose_1	patternClose_2	patternClose_3
r01	0.805270667	0.765840063	0.717410675	0.7417019	0.7279593	0.8109946
r02	0.805676	0.767868946	0.701623008	0.7393902	0.7202731	0.80727
r03	0.870232667	0.82447205	0.748077105	0.7498833	0.7184068	0.8347232
r04	0.959663	0.895639864	0.791558482	0.7420557	0.7098779	0.8370837
	0.81672575	0.80008035	0.730129098	0.7418354	0.7458664	0.8191963
	0.800927	0.779110901	0.690255185	0.7361524	0.7365715	0.8060915
	0.867331	0.835950436	0.737223165	0.7417046	0.7355225	0.832611
	0.95778775	0.906068373	0.791928194	0.7348577	0.7259914	0.8356983
	0.715774667	0.705791743	0.564368194	0.7305425	0.7330422	0.7839939
	0.730252333	0.72947828	0.58001116	0.7355208	0.729423	0.7894829
	0.83146	0.80805	0.647614237	0.7474076	0.739249	0.8254348

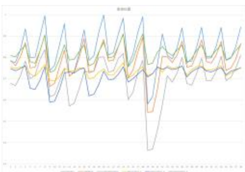
	0667	9956				
	0.9378325	0.889498371	0.716311063	0.7346453	0.7303093	0.8394462
	0.814047	0.79618871	0.72390209	0.7497772	0.7492616	0.819696
	0.813456	0.804502819	0.70810666	0.7483939	0.7436139	0.81828
	0.871534	0.848053374	0.75843912	0.7550951	0.7431798	0.841826
	0.957381	0.905886518	0.80988009	0.743084	0.732405	0.846802
	0.794785	0.766441814	0.69555432	0.7160926	0.7225716	0.795393
	0.800402	0.784484066	0.67607724	0.7336204	0.7331367	0.802817
	0.866701	0.83920819	0.73183885	0.7493189	0.7367068	0.832814
	0.958516	0.905634855	0.79288904	0.7449184	0.7311733	0.844377
	0.797833	0.781656	0.64550927	0.7324109	0.732769	0.799945
	0.815219	0.808259234	0.66524412	0.744951	0.7428843	0.815357
	0.879946	0.859006274	0.72162454	0.7432307	0.7353125	0.83545
	0.968336	0.907721793	0.78049901	0.7230855	0.6972772	0.828626
	0.803566	0.767749693	0.71651166	0.7209006	0.7235456	0.79494
	0.812609	0.789425734	0.71785945	0.7333797	0.7312254	0.808065

	0.872075	0.840681606	0.75589657	0.7407426	0.7370709	0.836057
	0.961187	0.906646683	0.80970424	0.7426951	0.727943	0.84628
	0.831922	0.792579666	0.75445923	0.7267558	0.7328684	0.820588
	0.816696	0.789632735	0.72211647	0.7343377	0.7312403	0.811156
	0.876335	0.841048751	0.76590759	0.7416666	0.7380583	0.836369
	0.961475	0.905814318	0.8108972	0.7389165	0.7239349	0.842291
	0.80984	0.7878167	0.71190105	0.7497745	0.7309429	0.810227
	0.805537	0.788668201	0.69987134	0.7506921	0.7284788	0.8087
	0.870163	0.842496973	0.74877296	0.7534789	0.7304837	0.837316
	0.95848	0.903558998	0.8066449	0.7515786	0.7250603	0.844643
Total	30.816975	29.67102304	26.14661658	26.64459	26.31364	29.60004

Solid



dense



sparse

scant

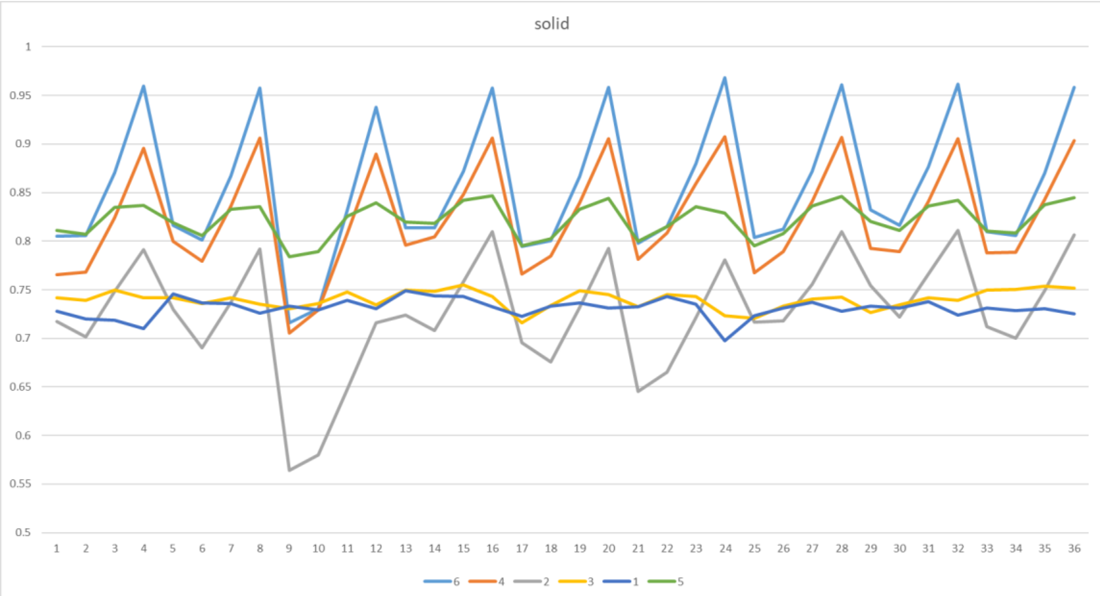
改进后顺序

按照熵值越小，上下文效果越好的评判标准，我们可以得到较为理想的上下文编码顺序为：

patternClose_2>>orderedPclosePar>>patternClose_1>>neighbEnd>>patternClose_3>>direction

```
//<<sky add
int ctxMap2=(patternClose & (0b00011000))<< 13-3;

int orderedPclosePar = (((pattern >> 5) & 3) << 2) + (!!(pattern & 128) << 1) + !!(pattern & 256);
ctxMap2 |= orderedPclosePar<<9;
ctxMap2 |= (patternClose & (0b00000110)) << 7 - 1 ;
ctxMap2 |= neighbEnd << 3;
ctxMap2 |= (patternClose & (0b00000001))<< 2;
ctxMap2 |= direction;
```



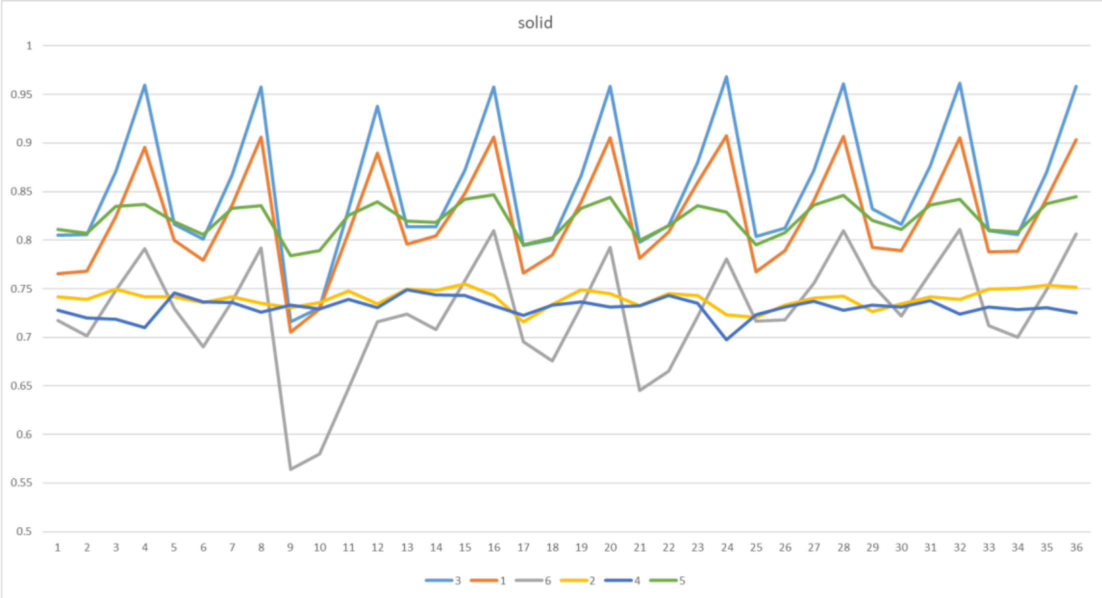
性能结果

C2_ai	lossy geometry, lossy attributes [all intra] End-to-End BD-AttrRate [%] Luma	Chroma Cb	Chroma Cr	Reflectance	Geom. BD-TotGeomRate [%] D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.5%	0.5%
Dense average	0.0%	0.0%	0.0%	0.0%	0.4%	0.4%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%
Scant average	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.2%	0.2%

四种类型数据有 loss，与理论上不符合

观察原始的次要信息使用顺序，发现他将熵值较大的和较小的交错使用，推测是否由于上下文之间的相关性对效果有影响。

现有上下文顺序在 solid 类型熵值曲线图中的体现：



改进方案二

结合原始的上下文排列顺序以及方案一过程中测得的熵值，尝试将上下文以如下顺序排列：

改进后顺序

```
int ctxMap2 = (patternClose & (0b00000110)) << 13 - 1;
ctxMap2 |= neighbEnd << 9;
ctxMap2 |= (patternClose & (0b00011000)) << 7 - 3;
ctxMap2 |= direction << 5;
int orderedPclosePar = (((pattern >> 5) & 3) << 2)+ (!!(pattern & 128) << 1) + !!(pattern & 256);
ctxMap2 |= orderedPclosePar << 1;
ctxMap2 |= (patternClose & (0b00000001));
```

性能结果

C2_ai	lossy geometry, lossy attributes [all intra] End-to-End BD-AttrRate [%] Luma	Chroma Cb	Chroma Cr	Reflectance	Geom. BD-TotGeomRate [%] D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%
Dense average	0.0%	0.0%	0.0%	0.0%	0.2%	0.2%
Sparse average	0.0%	0.0%	0.0%	0.0%	-0.1%	-0.1%
Scant average	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.0%	0.0%

编码位置信息用到的次要信息

- *missedCloseStart*：四条垂直边中不包含顶点的数量
- *patternClosest*：表示 9 条边中在 closeest 区间被占据的情况
- *direction*：当前待编码边的朝向（0=X，1=Y，2=Z）

- *patternClose*：表示 9 条边中在 close 区间被占据的情况

现有顺序

```
// first bit
ctxMap1 = ctxFullNbounds * 2 + (nclosestStart > 0);
ctxMap2 = missedCloseStart << 8;
ctxMap2 |= (patternClosest & 1) << 7;
ctxMap2 |= direction << 5;
ctxMap2 |= patternClose & (0b00011111);
int orderedPclosePar = (((patternClose >> 5) & 3) << 2) + (!!(patternClose & 128) << 1) + !!(patternClose & 256);

int bit = (vertex >> b--) & 1;
arithmeticEncoder->encode(bit, ctxTriSoup[MapOBUFTriSoup[1].getEvolve(bit, ctxMap2, ctxMap1)]);
v = bit;

// second bit
if (b >= 0) {
  ctxMap1 = ctxFullNbounds * 2 + (nclosestStart > 0);
  ctxMap2 = missedCloseStart << 8;
  ctxMap2 |= (patternClose & 1) << 7;//为什么单独将这个提前？？？？？
  ctxMap2 |= (patternClosest & 1) << 6;
  ctxMap2 |= direction << 4;
  ctxMap2 |= (patternClose & (0b00011111)) >> 1;
  ctxMap2 = (ctxMap2 << 4) + orderedPclosePar;

  bit = (vertex >> b--) & 1;
  arithmeticEncoder->encode(bit, ctxTriSoup[MapOBUFTriSoup[2].getEvolve(bit, ctxMap2, (ctxMap1 << 1) + v)]);
  v = (v << 1) | bit;//00 01 10 11,二值化以后的顶点位置坐标信息
}
```

改进方案一

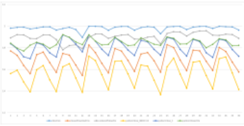
solid 类型序列测试结果

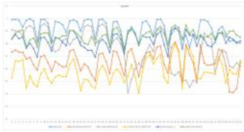
name	direction	missedCloseStartCtx	orderedPclosePar	patternClose_00011111	patternClose_1	patternClosestCtx
r01	1.980882	1.770277483	1.891980477	1.561573493	1.837333467	1.841771817
r02	1.988420667	1.7294187	1.921497991	1.593087376	1.790419533	1.796192917
r03	1.9906405	1.636597233	1.919145113	1.483829337	1.716792333	1.769146333
r04	1.973973333	1.55976535	1.886510369	1.392038738	1.698755167	1.823811167
	1.981973	1.735981075	1.88435577	1.596963643	1.84571725	1.850236
	1.99282125	1.7597555	1.918854843	1.635146137	1.82503125	1.83281525

	1.99342825	1.662250125	1.91788387	1.503471695	1.75281225	1.7940035
	1.9632095	1.571907775	1.875832718	1.394575185	1.7144875	1.82772075
	1.977851833	1.74778723	1.780680942	1.62399531	1.919315833	1.921621333
	1.989368167	1.731798183	1.820153525	1.652318659	1.8957415	1.897230167
	1.967749833	1.639199867	1.827674405	1.520522679	1.834313167	1.853617833
	1.895268833	1.543414202	1.791546672	1.38758011	1.762192333	1.828887667
	1.998076	1.8260541	1.89539177	1.72282447	1.919322	1.920016
	1.998121	1.754916	1.92654124	1.68147038	1.860998	1.861751
	1.995789	1.6552752	1.925995787	1.55065919	1.785958	1.825486
	1.961326	1.5673259	1.886892205	1.41116828	1.728668	1.831954
	1.990843	1.793421	1.90595991	1.6740613	1.892976	1.897052
	1.996804	1.769073	1.9172706	1.677714358	1.849734	1.857692
	1.996541	1.683352	1.92267686	1.55950857	1.779707	1.819561
	1.962254	1.5807559	1.880404516	1.4253628	1.72869	1.826341
	1.986607	1.7413258	1.87806908	1.64077665	1.867938	1.891962
	1.996001	1.755817	1.897352	1.6302101	1.854878	1.860874

			407	1		
	1.990694	1.679787	1.906313162	1.49989211	1.785267	1.833158
	1.929211	1.57329825	1.83571289	1.36620815	1.730547	1.822042
	1.99312	1.828398	1.93189146	1.62412491	1.896597	1.900127
	1.996243	1.8025	1.94815278	1.70460638	1.88008	1.887369
	1.99744	1.6866719	1.945816906	1.55606403	1.785448	1.834276
	1.967958	1.5878979	1.908303263	1.42518285	1.722824	1.835503
	1.991399	1.778342	1.94325113	1.66637891	1.876931	1.882273
	1.996904	1.771122	1.95505382	1.67632044	1.857308	1.864147
	1.997255	1.674604	1.95709444	1.54298498	1.753803	1.799747
	1.979015	1.5901114	1.916049244	1.41872167	1.711033	1.831677
	1.998183	1.802846	1.91382096	1.69895721	1.879661	1.883164
	1.994816	1.765749	1.92561095	1.71465149	1.86617	1.873804
	1.992785	1.6672727	1.922212681	1.56692778	1.783964	1.818344
	1.963231	1.5762434	1.88740058	1.41343522	1.721285	1.821911
total	71.36620317	61.00031217	68.36935534	56.1933146	65.11269858	66.51728573

solid dense sparse scant





改进后顺序

按照熵值越小，上下文效果越好的评判标准，我们可以得到较为理想的上下文编码顺序为:

patternClose&0b00011111>>missedCloseStartCtx>>orderedPclosePar>>patternClose&1>>patternClosestCtx>>direction

性能结果

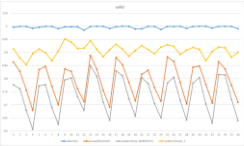
	name		anchor	test
solid	dancer_vox11_00000001	r04	582696	608848
		r01	19964	19968
	basketball_player_vox11_00000200	r04	666712	691968
		r01	22648	22656
dense	boxer_viewdep_vox12	r01	27440	27472
		r04	803504	8563688
scant	ulb_unicon_vox20	r01	71384	71848
		r04	285096	288064

测试的几个序列都是 loss

position 的两个 bit 位置信息不能用同一套上下文顺序，应该分开考虑

改进方案二——first bit

分开统计 2bit 位置信息不同上下的熵值，按其原有种类，重新按熵增排序，寻求增益。

Solid	Dense	Sparse	Scant
			

改进后顺序

patternClose&0b00011111>>missedClosePar>>patterClosest&1>>direction

改进方案二——second bit

Solid	Dense	Sparse	Scant
			

改进后顺序

patternClose&0b00011111>>missedClosePar>>patternClose&1>>orderedPclosePar>>patternclosest&1>>direction

性能结果

C2_ai	lossy geometry, lossy attributes [all intra] End-to-End BD-AttrRate [%] Luma	Chroma Cb	Chroma Cr	Reflectance	Geom. BD-TotGeomRate [%] D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.9%	1.0%
Dense average	0.0%	0.0%	0.0%	0.0%	0.4%	0.5%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%
Scant average	0.0%	0.0%	0.0%	0.0%	0.2%	0.2%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.4%	0.4%

1. position 的两个 bit 位置信息不能用同一套上下文顺序，应该分开考虑

标识次要信息编码优化+位置次要信息编码优化性能结果

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	1.0%	1.0%
Dense average	0.0%	0.0%	0.0%	0.0%	0.6%	0.6%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%
Scant average	0.0%	0.0%	0.0%	0.0%	0.2%	0.2%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.4%	0.5%

问题分析与改进思路

1. 观察编码标识信息的次要信息使用顺序，发现他将熵值较大的和较小的交错使用，推测是否由于上下文之间的相关性对效果有影响。

2. 从标识信息的第二种改进方法可以看出，当结合现有的次要信息使用顺序以及熵值越小上下文越有效的结论以后，性能相比完全按熵增排序要好，甚至在 sparse 类型有 0.1%增益。

3. 位置信息分开来按照熵增排序以后使用，得到和标识信息一样的结果，也是与理论不符合，有 loss。考虑是否也跟相关性有关？