Student number	point total	req total	extra total	R1 Bezier (2p)	R2 B-spline (2p)	R3 Gen triangles (2p)	R4 new positions (2p)	R5 old positions (2p)	mod	notes / comments / ...	boundary handling (3p)	local coordinate frames (1p)	surfaces of revolution (3p)	gencyls (3p)	new subdiv schemes (?p)	camera path (4p)	other (put points here)	what other extras?
218096	0	0	0															
225157	0	0	0															
270034	1.5	1.5	0	1.5						R1: Transpose your basis matrix and the code will work								
292009	4	4	0	2	2													
292326	13	10	3	2	2	2	2	2			3							
292986	0	0	0															
293545	4.5	4.5	0	2	2	0.5												
295323	0	0	0															
297606	8.5	8.5	0	2	2	2	2	0.5		R5: You have the right idea.								
311210	0	0	0															
347022	0	0	0															
350006	16.5	10	6.5	2	2	2	2	2		R5: B should be 3 / 16, when n == 3. Also, generally normals should be updated similarly to positions. This difference can be seen in patch.swp (lighting shows faint "stripes"); Local coordinate frames: Binit is being reset between curve segments, which causes the discontinuities. This is why sponza.swp acts so weirdly;	3	0.5	3					
350475	2	2	0	2														
353692	0	0	0															
353757	9.5	8.5	1	2	1.5	2	2	1		R2: For Bezier, you need 3n+1 control points but you output 4n. See weirder.swp; R5: Correct idea and structured reasonably, the one-ring loop doesn't work.							1	Subdiv colors(1p)
357083	0	0	0															
362256	0	0	0															
401311	0	0	0															
424615	9.5	9.5	0	2	1.5	2	2	2		R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp.								
425494	0	0	0															
425614	9	9	0	2	2	1.5	2	1.5		R3: indeed, "odd" has a different winding order than the rest of the triangles. R5: n should begin at 0.								
426419	0	0	0															
426736	0	0	0															
427492	0	0	0															
427793	8.5	8.5	0	2	1.5	1.5	2	1.5		R2: The loop condition and increment aren't correct. Check again from slides how the bspline sliding window should work; R3: The triangle winding order isn't correct; R5: The first vertex is incorrectly counted twice in the ring								
427845	0	0	0															
428381	0	0	0															
428789	10	9.5	0.5	2	2	2	2	1.5		R5: should iterate until come back to the original vertex (an easy fix would be do{...}while(next_i!=i);), Boundary handling non-smooth	0.5							
430324	0	0	0															
430463	10	10	0	2	2	2	2	2										
431857	9	9	0	2	2	2	2	1		R5: The second loop is not necessary, line 260 fix the triangle index and the subdiv should work.								
432241	7.5	7.5	0	2	2	2	1.5			R4: Just missing a couple of %3 from the indexing								
437631	0	0	0															
438397	0	0	0															
472379	0	0	0															
473158	0	0	0															
473420	0	0	0															
473637	0	0	0															
474380	2	2	0	2						Very close; i and steps are both integral variables so i / steps performs an integer division that then gets casted to a float. If you do t = i/float(steps), the Bezier works.								
475389	1	1	0	1						R1: Does not work for multiple segments, and your Bezier matrix has an incorrect column 2.								
475758	0	0	0															
475910	0	0	0															
476498	0	0	0															
477170	0	0	0															
477400	0	0	0															
477617	0	0	0															
477659	6	6	0	2	1.5	1.5	1			R2: In longer curves, you have duplicate control points when one segment ends and one starts. The vector with new control points should be of length 3n+1. Now it is simply 4n; R3: new_vertices[edge] should contain the index to the newly creted vertex, not to one of the old ones (v0 or v1). The new vertex index is new_positions.size()-1. To see the subdivision you should have commented out the code at the end of the function; R4: The third sum in pos/col/norm should not be positions[i][(j+2)%3] but positions[indices[i][(j+2)%3]] Local coordinate frames: Not correctly calculated;		0						
477701	0	0	0															
478328	0	0	0															
478470	4	4	0	2	2													
478687	0	0	0															
479505	0	0	0															
479576	13.5	10	3.5	2	2	2	2	2		Local coordinate frames: DB has -3 as the first element of the second row; should be 3, BI is not stored to C.B and Binit is not passed to subsequent calls (and bezier has a Binit of 0 to begin with; this leads to zero B and N for the whole spline)	3	0.5						
479725	13	10	3	2	2	2	2	2			3							
480248	0	0	0															
480303	0	0	0															
480730	38	10	28	2	2	2	2	2			3	1.5	3	3	4	4	9.5	Local coordinate frames smooth over loops (+.5), adaptive step size (4p), Kochanek-Bartels splines (3p), Textures and displacement (2p)
481014	10	10	0	2	2	2	2	2										
481441	7.5	7.5	0	2	2	1.5	2			R3: The definitions of the edges a, b and c are incorrect. Accessing new_vertices isn't necessary here.								
493578	0	0	0															
506355	8	8	0	2	1.5	2	2	0.5		R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp;								
508793	0	0	0															
514020	0	0	0															
516109	20	10	10	2	2	2	2	2			3	1	3	3				
519656	0	0	0															
525653	0	0	0															
525666	0	0	0							zip is empty								
525792	0	0	0															
525925	14.5	10	4.5	2	2	2	2	2		Weird formula for Binit, doesn't make frames continuous over spline segment boundaries. Should just use the B of the last vertex of the previous segment.	3	0.5					1	Subdiv colors (1p)

Student number	point total	req total	extra total	R1 Bezier (2p)	R2 B-spline (2p)	R3 Gen triangles (2p)	R4 new positions (2p)	R5 old positions (2p)	mod	notes / comments / ...	boundary handling (3p)	local coordinate frames (1p)	surfaces of revolution (3p)	gencyls (3p)	new subdiv schemes (?p)	camera path (4p)	other (put points here)	what other extras?
526490	13	10	3	2	2	2	2	2			3							
526717	12	10	2	2	2	2	2	2		Boundary: Neighbours contains only the first boundary vertex found because immediatly after while loop breaks (could add "|| found" to the condition to prevent this). Also, once this is fixed, the second vertex you add to neighbours is not correct. (j2 + 1) % 3 should be j2 in push_back.	2							
526746	0	0	0															
527143	0	0	0															
527347	4	4	0	2	2					R2: The idea was to convert the points to Bezier base and then call coreBezier, not have the basis matrix as function parameter.								
527389	0	0	0															
527444	0	0	0															
527923	13	10	3	2	2	2	2	2			3							
528634	7	7	0	2	1.5	2	1.5			R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp. R4: neighboredgeindex is the corresponding vertex/edge on the other side; this is the same vertex as v1. You need a (... +2)%3 to find the last vertex.								
528883	0	0	0															
529293	0	0	0															
529303	0	0	0															
529617	0	0	0															
529992	20	10	10	2	2	2	2	2		Local coordinate frame: A way to fix the discontinuity would have been to shift the frames at the end of evalBezier so that the frames at the beginning and the end of the loop match (this should be done only if curve is closed of course).	3	1	3	3				
530185	0	0	0															
530907	7.5	7.5	0	2	1.5	2	2			R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp;								
530981	3.5	3.5	0	1.5	2					R1: T vector incorrect, should be Vec4f(1,t,t^2,t^3)								
540094	0	0	0															
540311	13	10	3	2	2	2	2	2			3							
541543	8	8	0	2	2	2	2	0		R5: Not quite enough for partial credit								
544375	13	9.5	3.5	2	2	2	2	1.5		R5: In verts the first and last vertex are the same causing the new positions to be slightly skewed. This can be seen e.g. in the wireframe of the icosahedron. Local coordinate frames: Coordinate frames not continuous between spline segments. Binit is invalid when tang[2] == 0.	3	0.5						
544566	0	0	0															
549749	0	0	0															
552969	0	0	0															
556347	0	0	0															
561578	0	0	0															
563068	0	0	0															
570116	0	0	0															
586210	20.5	10	10.5	2	2	2	2	2		Are you sure you didn't compare to the quaternion camera (default in the example)? The tangent-oriented ones look quite similar	3	1.5	3	3				Local coordinate frames smooth over loops (+.5)
587170	19.5	10	9.5	2	2	2	2	2		Tangent not normalized, forcing Binit to be z axis only makes for quite peculiar results	3	0.5	3	3				
587921	22.5	10	12.5	2	2	2	2	2		Local coordinate frames: Coordinate frames not continuous between spline segments. You were not supposed to calculate the normals with the second derivative but use the formulas given on page 18 of the assignment 2 handout. Adaptive step size: Error > 0 crashes code because begin and end become 0. Change conditions on row 229. You could compare derivatives (tangents) at begin and end and check that end and begin are further than minstep away from each other.	3	0.5	3	3			3	Subdiv colors (1p); Adaptive step size (2p)
588137	0	0	0															
588441	8.5	8.5	0	2	0.5	2	2	2		R2: You have to change the basis of your control points from bspline to bezier before you can evaluate the curve using evalBezier. Check the lecture slides to see how. In addition, you are not transfering all points to R (compare inner loop to similar one in evalBezier).								
589291	0	0	0															
589437	10	10	0	2	2	2	2	2		R4&R5: Generally normals should be updated similarly to positions.								
589848	0	0	0															
590112	9.5	9.5	0	2	1.5	2	2	2		R2: The curve is missing some points from the longer curves, check the curves in gcyl folder to see this.								
590332	17.5	8.5	9	2	2	1.5	2	1		R3: Incorrect triangle winding order; R5: The loop doesn't go through all the relevant vertices; Your coordinate frames are not orthonormal, your binormal should be orthogonal to the tangent, now it is always pointing at (0,1,0).		0	3	3			3	CR-Splines(3p)
590426	19.5	10	9.5	2	2	2	2	2		Local coordinate frames not continuous between segments; Srev: Isn't quite as smooth as the example because you're not calculating analytical tangents for the local coordinate frames (this is actually very simple, just differentiate the basis vector!); Gcyl: Your triangles are facing inwards and the closer faces are therefore getting culled. You can quickly fix this by switching the sweep loop direction and inverting the normals; Camera path: Your quaternion to matrix formula goes wrong because we define the quaternion q = qw+i*qx+j*qy+k*qz as Vec4(qx,qy,qz,qw). You also need to map t somehow. The given t is for the whole camera path, not only the current segment.		0.5	3	2		3	1	Subdiv colors(1p)
593177	0	0	0															
593452	10	10	0	2	2	2	2	2		R2: The idea was to convert the points to Bezier base and then call coreBezier, not have the basis matrix as a function parameter.								
593876	13	10	3	2	2	2	2	2			3							
594367	6	6	0	2	2	2												
594590	1.5	1.5	0	1.5						Unnecessary complexity in evalBezier. You can rewrite this in a single quite simple loop where you loop through the control points and append the results of coreBezier to your total curve.								
594930	0	0	0															
595201	21	10	11	2	2	2	2	2		Coordinate frames not continuous over the whole curve, this also causes the issue with gcyl;	3	1				4	3	Subdiv colors(1p), Visualizing curvature (2p)
595612	11	10	1	2	2	2	2	2									1	Subdiv colors (1p)
596048	14.5	10	4.5	2	2	2	2	2		Local coordinate frames: calling evalBezier separately for each patch resets Binit between spline segments.	3	0.5					1	Subdiv colors (1p)
596242	13	10	3	2	2	2	2	2			3							
596747	0.5	0.5	0	1					-0.5	Complier errors (-0.5p); R1: In coreBezier, t is either 0 or 1 because in C++ dividing integers gives you an integer (convert either i or steps to float). You are also returning an empty curve; In evalBezier, the loop stops before the last spline segment. Local coordinate frames: Normal not perpendicular to tangent or binormal. You were supposed to use the formulas on page 18 of assigment 2 to ensure that frames are calculated correctly.		0						
596789	0	0	0															
596792	11.5	10	1.5	2	2	2	2	2		Local coordinate frames: Not continuous over spline segments for B-splines because you call evalBezier for only one segment. This can be seen in sponza.swp.		0.5					1	Subdiv colors(1p)
596857	2.5	2.5	0	2	0.5					Next time please return all the files, not just the ones in the src directory, so that the code can be easily compiled. R1&R2: Using matrices and vectors would have made the computations a lot easier. R2: You were supposed to convert the new points to Bezier base before calling evalBezier.								
597445	10	10	0	2	2	2	2	2		R2: The idea was to convert the points to Bezier base and then call coreBezier, not add a new parameter for coreBezier.								
598088	10	10	0	2	2	2	2	2										

Student number	point total	req total	extra total	R1 Bezier (2p)	R2 B-spline (2p)	R3 Gen triangles (2p)	R4 new positions (2p)	R5 old positions (2p)	mod	notes / comments / ...	boundary handling (3p)	local coordinate frames (1p)	surfaces of revolution (3p)	gencyls (3p)	new subdiv schemes (?p)	camera path (4p)	other (put points here)	what other extras?
598318	0	0	0															
602851	0	0	0															
603067	8	8	0	2	1.5	2	2	0.5		R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp;								
603096	6	6	0	2	2	2												
603326	0	0	0															
604105	12.5	10	2.5	2	2	2	2	2		Boundary handling: should be done based on two neighboring vertices on the boundary; the weights are given in the second link of the extra description (and page 70 of the course notes)	1.5						1	Subdiv colors (1p)
606064	0	0	0															
606268	2	2	0	2						Dividing the steps isn't necessary here. The intended use is that this is the number of steps per segment so that longer curves also retain the same resolution.								
608952	0	0	0															
609142	20	10	10	2	2	2	2	2		Surface of revolution: Normals should also be rotated to get correct lighting. Normals and binormals in wrong columns. If their positions are switched, then the triangles face the wrong direction. (The normals need to be inverted. Like the handout says: With this convention, you will actually want to reverse the orientation of the curve normals when you are applying them to the surface of revolution.)	3	1	3	2			1	Subdiv colors(1p)
609155	0	0	0															
609168	13	10	3	2	2	2	2	2			3							
610827	2.5	2.5	0	2	0.5													
612155	0	0	0															
612540	0	0	0															
612812	0	0	0															
621308	0	0	0															
647175	0	0	0															
647502	17	10	7	2	2	2	2	2		Binit truly is arbitrary; it just chooses the original orientation of the basis (you could rotate all of the coordinate frames around their tangents by an angle and get another valid set of frames). Good point on the surface of revolution and triangle winding order. What makes the case somewhat unintuitive is that our splines' vertex indices increase downward, whereas the explanation in the comment has control points in the opposite order; this introduces an extra flip for the winding. We'll do something about this for the future.	3	1	3					
648080	14	10	4	2	2	2	2	2		Local coordinate frames: Binit is being reset between curve segments, which causes the discontinuities. This is why sponza.swp acts so weirdly; Subdiv colors: the color of the new vertex depends on the surrounding vertices and thus all colors approach red which makes it difficult to see the age of a vertex.	3	0.5					0.5	Subdiv colors(0.5p)
648569	9.5	9.5	0	2	2	2	2	1.5		R5: v should initially be indices[i][(j + 2) % 3], not indices[i][(j + 1) % 3]. Now the result is a bit skewed which can be seen in the wireframe of the icosahedron.								
648860	0	0	0															
649458	0	0	0															
650191	0	0	0															
650560	1.5	1.5	0	1.5						R1: Due to integer division, i / step is either 0 or 1. Convert one of them to float. You're also using .push_back() on a vector that has already been initialized to steps+1 size so your end result will have steps+1 points in the origin before the actual curve starts;								
650829	10	10	0	2	2	2	2	2		R2: The idea was to convert the points to Bezier base and then call coreBezier, not to have a separate core function for B-splines.								
651640	13	10	3	2	2	2	2	2		Normals handled non-smoothly at boundary (doesn't affect the scoring, just looks a bit weird)	3							
651802	0	0	0															
652209	18.5	10	8.5	2	2	2	2	2		No sign of another subdivision scheme. In generalized cylinders, the rotation matrix should just be the (N,B,T) frame of the current sweep point. Binit is not updated to the last B of the previous curve, and thus the frames are not continuous.	3	0.5	3	2				
652584	27.5	10	17.5	2	2	2	2	2		Quaternions in the code are defined a bit differently than in the lecture slides. In the slides, they are defined (θ,v) but in the code as (v,θ). I don't know if this is explicitly stated anywhere but it could have been deduced from the functions in parse.cpp.	3	1.5	3	3		3	4	Subdiv colors (1p), Local coordinate frames smooth over loops (+.5)
653156	9	9	0	2	2	2	2	1		R5: The 1-ring is not correct. To make it work, when initializing nt and ne, index (j+2)%3 should be used (not j), n should only be incremented once in the loop, and pos (and norm+col) should be initialized with the position of indices[i][(j+2)%3]; Local coordinate frames: Read pages 17-19 in assignment 2 to see what went wrong (normals shouldn't be calculated with second derivative);		0						
653347	13	10	3	2	2	2	2	2		Don't worry, Binit is only used in extras.	3							
654142	13	10	3	2	2	2	2	2			3							
654294	10	10	0	2	2	2	2	2										
654618	14	10	4	2	2	2	2	2			3						1	Subdiv colors (1p)
655109	0	0	0															
655361	0	0	0															
655390	0	0	0															
656014	0	0	0															
657068	6	6	0	2	2	2												
657181	10	10	0	2	2	2	2	2										
657437	14	10	4	2	2	2	2	2			3						1	Subdiv colors(1p)
657767	33	10	23	2	2	2	2	2		Coordinate frames not continuous over the whole curve, this also causes the issue with gcyl;	3	1	3	3		4	9	Subdiv colors(1p), CR-Splines(3p), Isosurface extraction (5p)
657893	0	0	0															
663434	0	0	0															
665173	10	10	0	2	2	2	2	2										
665678	33	10	23	2	2	2	2	2		Coordinate frames not continuous over the whole curve, this also causes your issue with gcyl.	3	1	3	3		4	9	Subdiv colors(1p), CR-Splines(3p), Isosurface extraction (5p)
666208	10	10	0	2	2	2	2	2										
666211	9.5	9.5	0	2	2	2	2	1.5		n should begin at 0.								
666253	0	0	0															
710015	12	10	2	2	2	2	2	2		Boundary: Remove +1 from the line where you assign a value to new_j2, and inside if (new_i2 == -1) you should have (j+2), not (j+1).	2							
715298	12.5	9.5	3	2	2	2	2	1.5		R5: Do not add indices[i][(j + 2) % 3] into vertices initially because it is added later in the loop. Because this vertex is in the vector twice, the end result is a bit skewed which can be seen in the wireframe of the icosahedron. Normals should generally be updated similarly as positions.		3						
716734	0	0	0															
717377	0	0	0															
717539	0	0	0															
718020	0	0	0															
718208	19.5	10	9.5	2	2	2	2	2		When you're inserting the front element for the reverse search direction, you're using the local index of the triangle, (j+2)%3, instead of the global one, indices[i][(j+2)%3].	2.5	1	3	3				

Student number	point total	req total	extra total	R1 Bezier (2p)	R2 B-spline (2p)	R3 Gen triangles (2p)	R4 new positions (2p)	R5 old positions (2p)	mod	notes / comments / ...	boundary handling (3p)	local coordinate frames (1p)	surfaces of revolution (3p)	gencyl s (3p)	new subdiv schemes (?p)	camera path (4p)	other (put points here)	what other extras?
718512	9.5	8	1.5	2	1.5	2	2	0.5		R2: For Bezier, you need 3n+1 control points but you output 4n. See sponza.swp or weirder.swp. Coordinate frames: Binit should be set to the last B of the previous spline segment.		0.5					1	Subdiv colors (1p)
718826	21	10	11	2	2	2	2	2			3	1.5	3	3			0.5	Local coordinate frames smooth over loops (+.5), curvature visualization attempt (.5p)
719032	17	10	7	2	2	2	2	2		Local coordinate frames: Coordinate frames not continuous between spline segments; Subdiv colors: Color of vertex does not depend on age;	3	0.5	3				0.5	Subdiv colors(0.5p)
721619	4	4	0	2	2													
721923	0	0	0															
723154	0	0	0															
723329	14.5	8.5	6	1	2	2	2	1.5		R1: Infinite loop here, also curves longer than one segments do not work correctly; R5: The first vertex of the ring is counted twice; Subdiv colors: The idea here was to somehow visualize the subdivision process - for example by coloring the vertices based on their age; Coordinate frames: Some error in the matrix for calculating tangent (Hint: You can simply differentiate the basis vector and use the same bezier matrix for the tangent); Gcyl: Your triangles are facing inwards and the closer faces are therefore getting culled. You can quickly fix this by switching the sweep loop direction. There are also some vertices left at the origin;		0.5	3	2			0.5	Subdiv colors(0.5p)
723468	9.5	9.5	0	2	2	2	1.5	2		R4: 'v3' is incorrect; R5: Something causes the debug mode to get stuck here								
723484	18.5	9.5	9	1.5	2	2	2	2		R1: Your curves are missing a bit from the end, this is because you set t = i / (steps+1), it should be t = i / steps; R4 & Boundaries: You have an assignment '=' instead of comparison '==' inside the if statement when checking for boundaries for the new vertices; Coordinate frames not continuous between curve segments;	2.5	0.5	3	3				
723565	8.5	8.5	0	2	2	2	1.5	1		R4: the fourth vertex is wrong. Your commented-out code is very close, the correct position would be positions[indices[neighborTris[i][j]][(neighborEdges[i][j]+2)%3]]; we want the neighboring triangle and edge based on the current triangle and edge, and then we walk along the edges in the neighboring triangle. R5: the key idea of the algorithm seems correct								
723976	0	0	0															
724483	0	0	0															
726915	13	10	3	2	2	2	2	2		Local coord frames: You can simply differentiate (1,t,t^2,t^3) and multiply with the geometry and bezier basis to get the tangent. d(1,t,t^2,t^3)/dt = (0,1,2t,3t^2).	3							
728696	0	0	0															
729297	0	0	0															
732323	0	0	0															
737551	14	10	4	2	2	2	2	2			3	1						
765714	0	0	0															
765756	10.5	10	0.5	2	2	2	2	2		Reasonable boundaries	0.5							
765785	20	10	10	2	2	2	2	2		There is another obvious workaround to the Binit issue; just reset the global value at the begin of evalBspline. No points deduced; removing checkFlat is indeed suspicious but you knew what was wrong and how to fix it.	3	1	3	3				
765882	10	10	0	2	2	2	2	2		R4&R5: Generally normals should be updated similarly to positions.								
766108	0	0	0															
767136	0	0	0															
769396	0	0	0															
772419	6	5.5	0.5	2	1	1.5	1			R2: evalBspline just returns evalBezier with the original arguments(?), the b-spline construction pushes 4 elements for each point (should be 4 for one, usually first or last, and 3 for rest) and uses transposed reads to the geometry matrix G_. Local coordinate frames: Only the basis matrix or the basis vector should be differentiated (not both), your basis vector is incorrect (should be (0,1,2t,3t^2)), Binit should be last element but B has steps, not 4 elements. R3: first triangle has different winding, R4: neighborEdges gives the corresponding index on the other side; these don't match in general		0.5						
784465	0	0	0															
784847	0	0	0															
784902	0	0	0															
785053	0	0	0															
785134	15	10	5	2	2	2	2	2		Boundary handling isn't quite correct, it leaves the corners sharp. You can fix this by only using the else branch inside of if(flag) and using v2 = indices[tri][(edge+1)%3] instead of it being the original vertex [i][j].	2						3	Catmull-Rom(3p)
785163	0	0	0															
785228	0	0	0															
785257	6.5	6.5	0	2	2	2	0.5			R4: The default value of opositeVertex should be -1, and not 0, because 0 is a valid vertex number. Similarly vert[m] should be converted to -1, and not 0. v2 is incorrect, the i in the second index of indices should be j. You use neighbourTris.x in all of your conditions, which is why the code crashes when the object has borders. But using all of these conditions and loops is not necessary, you could have gotten the value of opositeVertex the same way as v2, you would have had to only figure out which edge you should use for indexing.								
785325	0	0	0															
785354	20.5	10	10.5	2	2	2	2	2		Coordinate frames not continuous over segments; Srev & Gcyl: Transform normals with the inverse transpose matrix (or rather, just the matrix itself since in this case it is just a pure rotation, i.e. the matrix is orthonormal and it's inverse equals it's transpose), normals are also inverted.	3	0.5	2	2			3	CR-Splines(3p)
785367	20.5	10	10.5	2	2	2	2	2		Coordinate frames not continuous over segments; Srev & Gcyl: Transform normals with the inverse transpose matrix (or rather, just the matrix itself since in this case it is just a pure rotation, i.e. the matrix is orthonormal and it's inverse equals it's transpose), normals are also inverted.	3	0.5	2	2			3	CR-Splines(3p)
785435	0	0	0															
785448	0	0	0	0														
785451	4	4	0	2	2													
785493	7.5	7.5	0	2	2	1.5	2			R3: The new vertex index should be size-1, not size (valid indices of arrays run from 0 to size-1) and you're not using the new indices (remove comments at the end of the method)								
785503	0	0	0															
785516	0	0	0															
795551	9	8	1	2	2	2	2	0									1	Subdiv colors(1p)
795577	7.5	7.5	0	2	2	2	1.5			R4: right_vertex_index should have (neighbor_edge_idx+2)%3 instead of 0 as the second index; now you essentially take a random index from the neighbor triangle.								
795593	4	4	0	2	2													
795629	0	0	0															
795658	0	0	0															
795674	12.5	9.5	3	2	2	2	2	1.5		R5: The loop is missing one vertex from the one ring;	3							
795713	9	9	0	2	1.5	2	2	1.5		R2: For Bezier, you need 3n+1 control points but you input 4n. See sponza. swp or weirder.swp; R5: Very inefficient!								
795865	15.5	10	5.5	2	2	2	2	2		For the boundary, when computing the old vertices you need to search the triangle fan into both directions in order to find the neighboring edge vertices; in your code, vertices.back() would be the first one. In local coordinate frames, BDeriv has a 6 on the lower right corner that should be a 3, and the normal and binormal are computed with elementwise product (a*b) instead of cross product (cross(a,b)). Slerp has a slight issue; acos always returns positive values, so you want to either check the dot product or if theta>pi/2, in which case we want to negate q1 or q2 (negating the dot product and changing theta' = pi-theta).	1.5	0.5				3.5		

Student number	point total	req total	extra total	R1 Bezier (2p)	R2 B-spline (2p)	R3 Gen triangles (2p)	R4 new positions (2p)	R5 old positions (2p)	mod	notes / comments / ...	boundary handling (3p)	local coordinate frames (1p)	surfaces of revolution (3p)	gencyl s (3p)	new subdiv schemes (?p)	camera path (4p)	other (put points here)	what other extras?
796178	10.5	10	0.5	2	2	2	2	2		Coordinate frames not continuous between spline segments, also rotated 90 degrees.		0.5						
798257	0	0	0															
801131	0	0	0															
804646	9.5	9.5	0	2	2	2	2	1.5		R5: v2 is added a second time before the loop exits, causing the result to be slightly off.								
807711	0	0	0	0.5					-0.5	No readme, no project files, doesn't compile (C++ has no operator ^ for floats), loops t to 2*pi instead of 1								
809609	0	0	0															
811383	0	0	0															
814872	0	0	0															
818315	0	0	0															
821289	0	0	0															
822709	0	0	0															
46596K	0	0	0															
55055P	0	0	0															
62727K	19.5	10	9.5	2	2	2	2	2		Coordinate frames not continuous between spline segments; should set Binit to the last B of the previous segment	3	0.5	3	3				
64879R	14	10	4	2	2	2	2	2		The commented lines that change Binit should in principle be there; see how they change sponza.swp for example.	3	1						
65451T	0	0	0															
67932J	1.5	1	0.5	1						coreBezier correct for positions. should normalize tangents and compute new binormal each iteration, Binit is just the initial value.		0.5						
69246M	13.5	10	3.5	2	2	2	2	2		Local coordinate frames: Bit of a hack solution. Would not recommend using global variables. binit does not change between bezier curve segments. In evalBspline, you could have just added all the control points into vector points before calling evalBezier and no global variable would have been needed. And because binit at the end of one curve may be the binit for the beginning of another, binit might be in the same direction as the curve's tangent.	3	0.5						
77241H	0	0	0															
77388B	1	1	0	0		1				R3: You got the X and Y correct in your code. The only thing left would have been to add to new_indices the 4 new triangles instead of the old one (even).								
83107B	0	0	0															
83854J	6	6	0	2	2	2												
84171B	4	4	0	2	2													
84805K	0	0	0															
k28342	0	0	0															
k90624	0	0	0															
k93517	0	0	0															