Student number	point total	req total	extra total	R1: joint positions (1p)	R2: joint rotations (2p)	R3: visualize joints (2p)	R4: skeletal SSD (4p)	R5: normal skinning (1p)	mod	notes / wtf / ...	GPU SSD (2p)	Wrist joint (5p)	IK (8p)	Animation (3p)	Dual quat (4p)	Other models (5p)	Pose capture (10p)	Style-based IK (10p)	Other extras (?p)	What other extras
218096	0	0	0																	
225157	0	0	0																	
270034	4.5	4.5	0	1	1.5	2				R2: picking the rotation axes from the matrix we're constructing leads to weird non-circular rotation paths. Instead of tbtb, you should pick the rotation axes from an identity matrix (just multiply them all together in one expression to do this easily; tbtb originally contains an identity.)										
292009	5	5	0	1	2	2														
292326	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
292986	0	0	0																	
293545	5	5	0	1	2	2														
295323	0	0	0																	
297606	9	9	0	1	2	2	4													
311210	0	0	0																	
347022	0	0	0																	
350006	14.5	9.5	5	1	2	2	4	0.5		R5&GPU SSD: Normals not normalized; Wrist: Unnatual stretching due to the way you recalculate your weights, most of the vertices between the elbow and the wrist should belong to the elbow. Your changes prevent opening skeletons with no wrists;	2	3								
350475	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
353692	0	0	0																	
353757	8.5	8.5	0	1	2	1	4	0.5		R3: You forgot to draw the bones; R5: normals transformed as positions, not normalized										
357083	0	0	0																	
362256	0	0	0																	
401311	0	0	0																	
424615	12	10	2	1	2	2	4	1			2									
425494	0	0	0																	
425614	12	10	2	1	2	2	4	1			2									
426419	0	0	0																	
426736	0	0	0																	
427492	0	0	0																	
427793	10	10	0	1	2	2	4	1												
427845	0	0	0																	
428381	0	0	0																	
428789	9.5	9.5	0	1	2	2	4	0.5		R5: The accumulator matrix N is initialized to identity by default, you want Mat4f N(.0f);, and normals are not normalized (and are transformed with w=1, leading to some pitch black areas).										
430324	0	0	0																	
430463	5	5	0	1	2	2														
431857	11.5	10	1.5	1	2	2	4	1		GPU SSD: Normals not normalized	1.5									
432241	9	9	0	1	2	2	4	0												
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	0	0	0																	
475389	0	0	0																	
475758	0	0	0																	
475910	0	0	0																	
476498	0	0	0																	
477170	0	0	0																	
477400	10	10	0	1	2	2	4	1												
477617	4	4	0	1	2	1				R3: Bones not visualized										
477659	9.5	9.5	0	1	2	2	4	0.5		R5: normals should be transformed with 0 in w component since they are directions, not positions. Also, normals not normalized										
477701	9	9	0	1	2	2	4													
478328	2	2	0	1	1					R2: You should have used the x-, y- and z-axis as the axes of rotation instead of joint.position.										
478470	5	5	0	1	2	2														
478687	0	0	0																	
479505	0	0	0																	
479576	0	0	0																	
479725	0	0	0																	
480248	0	0	0																	
480303	8	8	0	1	2	1	3.5	0.5		R3: The joint orientations aren't visualized correctly, instead they remain static. You should draw the lines in the direction of the orientation vectors instead of the standard xyz axes; R4&R5: SSD results aren't stored anywhere; R5: Normals not normalized										
480730	49	10	39	1	2	2	4	1			2	5	8		4	5			15	FBX Animation (1p), FABRIK IK(4p), Automatic weighting using proxy meshes (10p)
481014	8.5	8.5	0	1	2	1	4	0.5		R3: Orientation visualized incorrectly. The orientation vectors can be read from the columns of the matrix, not rows. R5: Normals not normalized										
481441	9.5	9.5	0	1	2	2	4	0.5		R4: The idea was to store the inverted toWorld matrices in computeToBindTransforms(). Now you're not really doing anything in this function since you do the inversion later; R5: Normals not normalized										
493578	0	0	0																	
506355	10	10	0	1	2	2	4	1												
508793	0	0	0																	
514020	0	0	0																	
516109	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)										
519656	0	0	0																	
525653	0	0	0																	
525666	9.5	9.5	0	1	2	2	4	0.5		R5: normals transformed as positions and not normalized (the whole model turns too bright when rotated)										
525792	0	0	0																	
525925	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2									
526490	11.5	9.5	2	1	2	2	4	0.5		R5&GPU SSD: Normals not normalized	2									
526717	9.5	9.5	0	1	2	2	4	0.5		R5: Normals should have a w-coordinate of 0 when doing homogeneous transform.										
526746	0	0	0																	
527143	0	0	0																	
527347	3	3	0	1	2															
527389	0	0	0																	
527444	0	0	0																	
527923	21.5	9	12.5	1	1	2	4	1		R2: euler_angles[0] should rotate the x-axis and euler_angles[1] the y-axis; GPU SSD&Dual quat: Normals not normalized because normalized vector is not saved into a variable; Wrist: Models with no wrist do not behave correctly after your code changes.	1.5	5			4				2	Dual quaternions on the GPU (2p)
528634	14.5	9.5	5	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases) and transformed with w=1. The GLSL compiler gives a weird error because you're trying to run code outside of a function (the shader is compiled as its own program, so anything directly inside FW_GL_SHADER_SOURCE is in the global scope of the shader), if you move your code inside main you'll see that you can't use the CPU-side variables skel_, ATTRIB_JOINTSi etc. You should instead use the shader inputs (the "layout(...) in ..." variables). Animation system halfway there, dual quaternions have some issues but a reasonable attempt.	1			2	2					
528883	0	0	0																	
529293	0	0	0																	
529303	0	0	0																	
529617	0	0	0																	

Student number	point total	req total	extra total	R1: joint positions (1p)	R2: joint rotations (2p)	R3: visualize joints (2p)	R4: skeletal SSD (4p)	R5: normal skinning (1p)	mod	notes / wtf / ...	GPU SSD (2p)	Wrist joint (5p)	IK (8p)	Animation (3p)	Dual quat (4p)	Other models (5p)	Pose capture (10p)	Style-based IK (10p)	Other extras (?p)	What other extras
529992	19	10	9	1	2	2	4	1			2	4		3						
530185	0	0	0																	
530907	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized and treated as positions (w should be 0)										
530981	0	0	0																	
540094	0	0	0																	
540311	9.5	9.5	0	1	2	2	4	0.5		R5: normal transformed as a position and not normalized										
541543	9	9	0	1	2	1.5	4	0.5		R3: You have used getRow when defining ahead when you should have used getCol; R5: Normals not normalized										
544375	10	10	0	1	2	2	4	1												
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	22	9.5	12.5	1	2	2	4	0.5		Your dual quaternion implementation is practically correct; GLSL treats matrices as arrays of column vectors stacked together, so the indices are transposed (column first, row second).	2	5			3.5				2	Dual quaternions on the GPU
587170	12	10	2	1	2	2	4	1			2									
587921	19.5	9	10.5	1	2	1.5	4	0.5		R3: Coordinate systems not scaled; R5: Normals not normalized; Dual quaternions: To get the shortest path of rotation, you should check the non-dual quaternions against each other and negate the weights for ones that point to a different direction.		5			3.5				2	Dual quaternions on the GPU (2p)
588137	0	0	0																	
588441	12	10	2	1	2	2	4	1			2									
589291	0	0	0																	
589437	9.5	9.5	0	1	2	2	4	0.5		R5: Incorrectly transforming with transpose matrix (also w should be 0). Normals not normalized.										
589848	0.5	0.5	0	0.5						R1 Visualization: You can get the joint position from the last column of the toWorld transformation matrix,										
590112	10.5	8.5	2	1	2	1	4	0.5		R3: You should use transform matrix columns as the coordinate vectors, not rows; R5&GRU SSD: Normals not normalized: GPU SSD: you don't initialize some variables in the shader, which results in random data as initial value on newer nvidia gpus due to driver differences. Also, there appears one out of bounds error (i-4 should be i when reading from aJoints1). No points reduced but good to keep in mind.	2									
590332	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
590426	14.5	9.5	5	1	2	2	4	0.5		R5: Normals not normalized and they are transformed as positions because getXYZ() is perfomed before multiplication. GPU SSD: Normals not normalized; Animation: It would look better if the if (nextPhase != 0) condition did not exist.	2			3						
593177	0	0	0																	
593452	9.5	9.5	0	1	2	2	4	0.5		R5: Incorrectly treating the normals as positions (w-coordinate 1). This causes the normals to have w ≠ 1 after transforming with the inverse transpose. ToCartesian() call then divides by the w-coordinate, giving incorrect results. Also, normals not normalized										
593876	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized										
594367	11.5	9.5	2	1	2	2	4	0.5		R5 & GPU SSD: Normals not normalized	2									
594590	5	5	0	1	2	2														
594930	0	0	0																	
595201	15	9.5	5.5	1	2	2	4	0.5		R5&GPU SSD: Normals should have w = 0; GPU SSD: Normals not normalized; Wrist: Weights could have been better. Hand seems sort of disconnected and sticks out when rotated. Maybe the wrist is too high up the arm.	1.5	4								
595612	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2									
596048	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2									
596242	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2									
596747	0	0	0																	
596789	0	0	0																	
596792	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
596857	8.5	8.5	0	1	2	2	3.5	0		R4:The sum matrix is incorrectly initialized to identity instead of zero matrix; R5: Normals not transformed at all										
597445	15	10	5	1	2	2	4	1			2	1		2						
598088	12.5	10	2.5	1	2	2	4	1		GPU SSD: The vertex shader loops 8 elements even though the inputs are vec4 (this is a limitation of the vertex attribute API), you should loop 4 elements and do the same thing for aJoints2/aWeights2.	1.5	1								
598318	0	0	0																	
602851	0	0	0																	
603067	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)										
603096	5	5	0	1	2	2				R1: Instead of looping through all joints and using getJointParent(i) to see if i is a child, you could have just looped through j.children.										
603326	0	0	0																	
604105	11	9.5	1.5	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases). The vertex shader loops 8 elements even though the inputs are vec4 (this is a limitation of the vertex attribute API), you should loop 4 elements and do the same thing for aJoints2/aWeights2 -- could also be the reason why this didn't work everywhere. To improve perf, remove the inverse transposes -- it's a no-operation for a rigid transformation.	1.5									
606064	0	0	0																	
606268	0	0	0																	
608952	9	9	0	1	1.5	2	4	0.5		R2: m12 and m21 are switched when extracting the rotation matrix, R5: normal is transformed with an implicit w=1 and not normalized										
609142	21.5	9.5	12	1	2	2	4	0.5		R5&GPU SSD: normals not normalized; Wrist joint: Some unnatural stretching. Wrist joints are at least slightly in a wrong position. IK: Finite differences not ideal speedwise. Method looks solid, results leave some room for improvement.	2	3	7							
609155	0	0	0																	
609168	11	9.5	1.5	1	2	2	4	0.5		R5&GPU SSD: Normals not normalized; GPU SSD: You're missing the remaining 4 weights from your calculation.	1.5									
610827	4	4	0	1	2	1				R3: instead of adding scale to a specific axis, you should add right, up or ahead times scale to joint_world_pos and draw the second endpoint there.										
612155	0	0	0																	
612540	0	0	0																	
612812	0	0	0																	
621308	0	0	0																	
647175	0	0	0																	
647502	12	10	2	1	2	2	4	1			2									
648080	9	9.5	0	1	2	2	4	0.5	-0.5	Compiler errors (-0.5p); R5: normals should be transformed with 0 in w component since they are directions, not positions, and they are not normalized										
648569	10	10	0	1	2	2	4	1												
648860	0	0	0																	
649458	0	0	0																	
650191	0	0	0																	
650560	0	0	0																	
650829	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
651640	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)										
651802	0	0	0																	
652209	25	10	15	1	2	2	4	1			2	5	2		4				2	Dual quaternions on the GPU
652584	25	10	15	1	2	2	4	1		Animation: Use atan2, not atan, when converting quaternions back to euler angles. atan can only give values between -pi/2 and pi/2, meaning angles outside this range are calculated incorrectly. (It would have probably been more intuitive to have the animation last longer when frame count increases.)	2	5		2	4				2	Dual quaternions on the GPU (2p)
653156	0	0	0																	
653347	12	10	2	1	2	2	4	1		GPU SSD: you don't initialize your vectors in the shader, which results in random data as initial value on newer nvidia gpus due to driver differences. No points reduced but good to keep in mind.	2									
654142	9.5	9.5	0	1	2	2	4	0.5		R5: Normal is not a point but a direction, i.e. w should be 0, and normals should be normalized										
654294	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										

Student number	point total	req total	extra total	R1: joint positions (1p)	R2: joint rotations (2p)	R3: visualize joints (2p)	R4: skeletal SSD (4p)	R5: normal skinning (1p)	mod	notes / wtf / ...	GPU SSD (2p)	Wrist joint (5p)	IK (8p)	Animation (3p)	Dual quat (4p)	Other models (5p)	Pose capture (10p)	Style-based IK (10p)	Other extras (?p)	What other extras
654618	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases). Normal w-component should also be 0 in principle; here the inverse transpose effectively sets it to 0 anyway.	2									
655109	0	0	0																	
655361	0	0	0																	
655390	0	0	0																	
656014	8	8	0	1	1.5	1	4	0.5		R2: Rotations were supposed to be calculated in setJointRotation; R3: You forgot to draw the bones; R5: Normals not normalized										
657068	8.5	8.5	0	1	2	2	3.5			R4: Mat4f initializes to identity by default. Since you're gathering a sum, you need to zero initialize the matrix first.										
657181	11.5	9.5	2	1	2	2	4	0.5		R5 & GPU SSD: Normals not normalized	2									
657437	9.5	9.5	0	1	2	2	4	0.5		R5: Normals transformed as positions and not normalized										
657767	29	9.5	19.5	1	2	2	4	0.5		R5&GPU SSD: Normals not normalized and treated as positions (w should be 0); Dual quaternions: To get the shortest path of rotation, you should check the non-dual quaternions against each other and negate the weights for ones that point to a different direction. IK: Ok attempt, the key parts seem to be there.	2	5	4	3	3.5				2	Dual quaternions on the GPU (2p)
657893	0	0	0																	
663434	0	0	0																	
665173	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)										
665678	26	9.5	16.5	1	2	2	4	0.5		R5&GPU SSD: normals transformed as positions and not normalized; Dual quaternions: To get the shortest path of rotation, you should check the non-dual quaternions against each other and negate the weights for ones that point to a different direction.	2	5		3	3.5	1			2	Dual quaternions on the GPU (2p)
666208	0	0	0																	
666211	10	10	0	1	2	2	4	1												
666253	0	0	0																	
710015	11.5	9.5	2	1	2	2	4	0.5		R5 & GPU SSD: Normals not normalized	2									
715298	12	10	2	1	2	2	4	1			2									
716734	0	0	0																	
717377	0	0	0																	
717539	0	0	0																	
718020	0	0	0																	
718208	14.5	9.5	5	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases). Nice animation system, but for a more practical solution you might want to consider only storing the keyframes and interpolating on the fly.	2			3						
718512	9.5	9.5	0	1	2	2	4	0.5		R5: normals should be normalized after the sum, not per every term.										
718826	19.5	9.5	10	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2	5		3						
719032	0	0	0							Submission zip doesn't open with standard Windows zip tool, 7zip opens it but it's almost empty.										
721619	0	0	0																	
721923	0	0	0																	
723154	0	0	0																	
723329	10.5	8.5	2	1	1	2	4	0.5		R2: On row 43 you are have Rx when you should have Rz. This causes the model to look flat when rotating around the x-axis. It would have been simpler to use the given Mat3f::rotation() function; R5: Normals transformed as positions. Your solution for GPU SSD is correct though.	2									
723468	11.5	8.5	3	1	2	1	4	0.5		R3: The columns of the transformation matrix would have given you the correct directions for the coordinate systems. Now the lines always point in the same direction. R5: Normals not normalized; Wrist: No weights and because left_hand is added in the middle of the file, the weights are off for some joints; GPU SSD: you don't initialize your vectors in the shader, which results in random data as initial value on newer nvidia gpus due to driver differences. No points reduced but good to keep in mind.	2	1								
723484	12	9.5	2.5	1	2	2	4	0.5		R5: Normals not normalized; GPU SSD: you don't initialize your matrix in the shader, which results in random data as initial value on newer nvidia gpus due to driver differences. No points reduced but good to keep in mind; Wrist: No weights and because left_wrist is added in the middle of the file, the weights are off for some joints. Wrists are not in the correct location.	2	0.5								
723565	9.5	9.5	0	1	1.5	2	4	1		R2: m11 is copied twice and m12 not at all, flattening the model on some rotations										
723976	0	0	0																	
724483	0	0	0																	
726915	10.5	9.5	1	1	2	2	4	0.5		R5: Normals not normalized; GPU SSD: The ssd code is exatly right apart from the second loop bounds. After the ssd, you should use the new normal for calculating clampedCosine and the new position for calculating gl_Position.	1									
728696	0	0	0																	
729297	0	0	0																	
732323	0	0	0																	
737551	12	10	2	1	2	2	4	1			2									
765714	0	0	0																	
765756	0	0	0																	
765785	20.5	10	10.5	1	2	2	4	1		Dual quaternions: To get the shortest path of rotation, you should check the real parts of the quaternions against each other and negate the weights for ones that point to a different direction.	2	5			3.5					
765882	7	7	0	1	2	2	2	0		R4: computeToBindTransforms and getSSDTransforms incorrect; R5: This should just be a matrix vector multiplication just as you did for positions.										
766108	0	0	0																	
767136	0	0	0																	
769396	0	0	0																	
772419	8.5	8.5	0	1	1	1.5	4	1		In R2 you're setting the wrong elements from to_parent; should be m3(i) instead of m(i)3 (or just zeros). This makes the model completely distorted under any joint movement. In R3, the last coordinates of right and up are read from the wrong columns; are 1 and 2 but should be 0 and 1.										
784465	0	0	0																	
784847	0	0	0																	
784902	0	0	0																	
785053	0	0	0																	
785134	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (sometimes overly dark in tight creases)										
785163	0	0	0																	
785228	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases). Good point about the key bindings, we'll do something about this.	2									
785257	11	9.5	1.5	1	2	2	4	0.5		R5: Don't apply translation to normals, i.e. set the w coordinate to 0 before transforming.The transposed() call doesn't do anything right now, use transpose() to modify the underlying object. Only transposing would give incorrect results, you need the inverse transpose. However in this case the transformation matrix is orthogonal which means it's inverse equals it's transpose and therefore the inverse transpose is just the original matrix. Also, normals not normalized; GPU SSD: Normals not transformed	1.5									
785325	0	0	0																	
785354	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
785367	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
785435	0	0	0																	
785448	0	0	0							Submitted but nothing done										
785451	5	5	0	1	2	2														
785493	6	6.5	0	0.5	2	1	3		-0.5	README.txt empty. R1: incorrect order of transformations (in the end we multiply with the position from the right; we go first into the parent bone's state and from there to the world), R3: the idea is to visualize the local coordinate system of the bone; instead of adding scale to a specific axis, you should add right, up or ahead times scale to joint_world_pos. R4: not zeroing the position between loops (you just keep on summing), unfortunate order of computations (do the matrix product in parentheses to get the correct answer.)										
785503	0	0	0																	
785516	0	0	0																	
795551	10	10	0	1	2	2	4	1												
795577	10	10	0	1	2	2	4	1												
795593	5	5	0	1	2	2														
795629	0	0	0																	
795658	0	0	0																	
795674	12	9.5	2.5	1	2	2	4	0.5		R5&GPU SSD: Normals not normalized; GPU SSD: Normals should have w = 0	1.5				1					

Student number	point total	req total	extra total	R1: joint positions (1p)	R2: joint rotations (2p)	R3: visualize joints (2p)	R4: skeletal SSD (4p)	R5: normal skinning (1p)	mod	notes / wtf / ...	GPU SSD (2p)	Wrist joint (5p)	IK (8p)	Animation (3p)	Dual quat (4p)	Other models (5p)	Pose capture (10p)	Style-based IK (10p)	Other extras (?p)	What other extras
795713	9	9	0	1	2	1	4	1		R3: E.g. for the x-axis you should have used joint_world_pos + scale * right										
795865	11.5	9.5	2	1	2	2	4	0.5		R5: normals not normalized (overly dark in tight creases)	2									
796178	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
798257	0	0	0																	
801131	0	0	0																	
804646	9.5	9.5	0	1	2	2	4	0.5		R5: normals not normalized (sometimes overly dark in tight creases)										
807711	0	0	0																	
809609	9.5	9.5	0	1	2	2	4	0.5		R5: Normals not normalized										
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	12	10	2	1	2	2	4	1			2									
64879R	9.5	9.5	0	1	2	2	4	0.5		R5: multiplying the weight into the matrix before inverting it works as if the matrix had a scaling component; the weight multiplication should be performed afterwards.										
65451T	0	0	0																	
67932J	4.5	4.5	0	1	1.5	2				R2: each rotation should simply happen against the corresponding axis.										
69246M	12	10	2	1	2	2	4	1			2									
77241H	0	0	0																	
77388B	0	0	0																	
83107B	0	0	0																	
83854J	5	5	0	1	2	2														
84171B	0	0	0																	
84805K	0	0	0																	
k28342	0	0	0																	
k90624	0	0	0																	
k93517	0	0	0																	