Multiscreen casting

> Imagine a boring night in the lab. Coding. Thinking. More coding. Fixing bugs. Introducing more bugs... Not so much fun is it? – What if we could go stand at the back of the room and turn all of these computer screens into one big screen, controlled by your smartphone? A countdown appears on the screens. You shake your phone and a cat appears on a nearby screen. You control the cat by making movements with your phone. Guide it through the maze of computer screens. Find the exit screen. Invite your friends and have a race, or find and chase a friend. It's ***Cat Caster*** time! – Of course, still one problem, this game hasn't been written yet...

Practical information

This year's P&O assignment consists of **developing a multiscreen casting framework for a heterogenous set of displays from different computers, tablets and smartphones, which have loaded an application webpage in their browser**. The camera on a master device (a smartphone or tablet) is used to discover the physical setup (location, orientation, size) of the slave devices from the point of view of a user. The application server allows for real time communication between all the clients.

You are assigned to a team of 6 students. We will use different channels to evaluate each team member, and allocate an individual score on your performance. To avoid teams being six headed monsters we assign two team members the special roles of **CEO** (chief executive officer) and **CTO** (chief technology officer). They stay in this role for the full year, unless the dynamics of a team force us to decide otherwise. (Please contact a member of the didactic team immediately in case of conflicts in your team.)

Your implementation will make use of a lot of different technologies. Most of these technologies will be new and require you and your team to go and investigate how things work. It is not sufficient to just produce code, you will have to demonstrate and convince us that the code actually works as promised. Convincing us is hard. **We require you to motivate choices and know how things work under the hood, and we definitely require you to test your solution in a scientific and reproducible way.** Your design, algorithms, decisions and tests will be part of the team report. Each team will be **required to use a git repository on GitHub** as a version control system. If in doubt, we will check commit logs to see if you contributed to the actual implementation. (We look for quality in your commits, not quantity. We expect commits on a regular basis since you are part of a team.)

To help you and your team to get on the right track, we organize two **technical committee meetings each week**. **TC1** will be on the topic of network communication and protocols, **TC2** will be on the topic of image analysis and algorithms. Each team needs to designate one person for each technical committee. In the second semester they will be replaced by other team members.

Technical committees

The technical committees will meet for one hour each week and the idea is to give all of the teams access to the same information. Sharing is good! Don't be afraid to share your good solutions! At least one member of the didactic team will follow each meeting, and will take note of particularly good and creative solutions.

Each committee assigns a chair for the full semester. The chair will make sure the committee starts and finishes on time (maximum 1 hour). The committee, under the supervision of the chair, can decide what to discuss, but the didactical team might steer the process. Immediately after the meeting, the chair needs to produce a short meeting report and post this on the discussion forum on toledo (as a plain message, not as an attachment), such that everybody has access. Teams can ask their representatives for more detailed information.

Each week between 3 and 4 students should give a **short (10 minutes) technical presentation** which will be evaluated by your peer committee members and a member of the didactical team. The chair has to make sure all committee members (except the chair) have given a presentation by the end of the semester. These

presentations should be posted to the message board on toledo in the form of a PDF (no other file formats!) by the presenter immediately after the TC meeting has finished.

An example of such a technical presentation for TC1 could be a tutorial on socket.io, with a focus on interesting and unexpected behaviour. E.g., showing the contents from arbitrary files when opening files by arguments passed in the URL (e.g., "../../script-with-login-credentials-to-database.js" or "/etc/passwd"). An example for TC2 might be how to calculate the principal symmetry axes of a blob of 2D pixels by using an inertia matrix and a 2-by-2 eigenvalue decomposition in closed form, and an example of when this could fail.

It should be clear that your presentation should pack as much content as possible in these 10 minutes. Do not waste time talking about trivial things which everybody knows from the assignment or giving too much background. **The chair should stick to the timing and limit the questions. Interesting questions, and follow up questions, are to be handled on the message board (available to everybody).**

In the second half of the semester the big technical problems should mostly have been discussed and the focus of the TCs needs to change to how to test the solutions. I.e., the presentations shift focus to testing.

TC1: Network communication and application framework

This technical committee will discuss problems and solutions for real-time communication between different browsers running on the clients, user interface design, resilience against communication problems, (web) security and developing strategies to test all this.

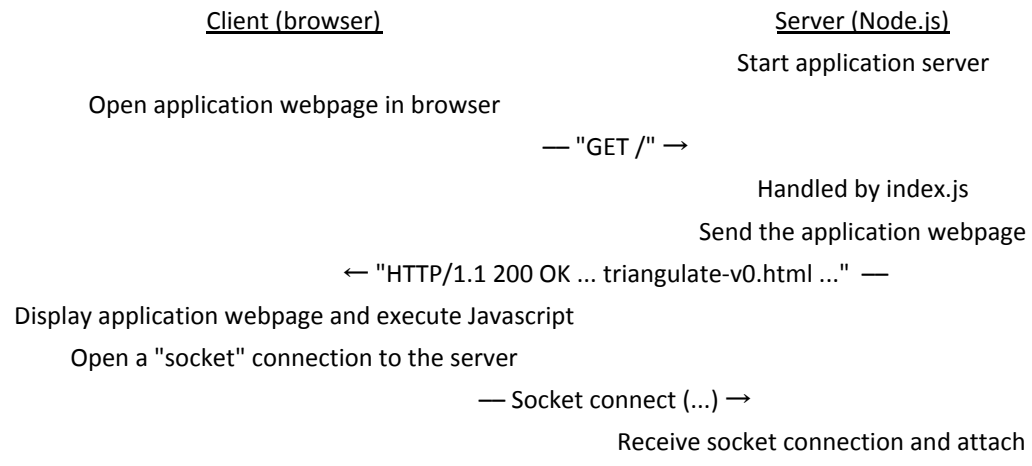
The application framework consists of the application server and the application webpage. The application server will be developed in Node.js which will serve a (single) HTML5 webpage to the client. The client will be a computer, smartphone or tablet with a modern browser.

The Node.js server can easily be run from your own computer, but for testing and demonstrations we require you to run your application server from a designated machine at the department (see toledo for instructions). Node.js is a Javascript framework. Javascript is probably new to most of you. You will have to invest some time to understand how to write Javascript code. It is also important to realise that **Javascript is single threaded by design**, and hence your server in Node.js will run as a single thread, also when multiple clients connect. This implies that all your blocks of Javascript code on Node.js should have a reasonably small execution time. We expect you to understand what it means that Node.js has asynchronous I/O with little blocks of Javascript code running single threaded whenever data is available.

The application page is a (single) HTML5 page. Writing (good) web pages and using the advanced HTML5 extensions (like accessing the camera) is probably also new. We give some pointers at the end of this section.

For the communication with the clients we need bi-directional real-time communication. Normal web traffic is served over HTTP, but this is not well suited for real-time communication. Therefore we will make use of a library called socket.io which is an easy way to use websockets (with some fallback options). There is also something called Webpush; we expect you to study and understand all of this terminology, and assess which features are needed.

A rough sketch of the initial part of the application framework goes as follows:



... ...

Note that it is easy to incorporate different application pages to be served. E.g., a different page for a different task in this assignment. It is also possible to keep explicit versions. (Please think this through in combination with git; git is not the solution to everything.) Take into consideration that other teams might also access your application server by using the URL and port. This can be used (ask other teams to test with their devices) and abused. Take appropriate measures. Note that it is **not** allowed to pass the filename of the application webpage as an argument, as you do not want a client to be able to access arbitrary files. Please take care of security and consult some best practice web security techniques! (This makes a perfect topic for a technical presentation.)

A client which connects becomes a slave device by default. There should be a way to turn a slave into a master. It should be clear by looking at the screen whether a device is a slave or a master. (Depending on the use case it might be that only one master is allowed, then you need to decide how to handle that.) A master will show a live camera feed in the browser window (you have to provide the option to switch to front or back camera) and and has the means to start the algorithm which will detect the slaves in the image. After the successful detection, the main application can be run (see the different tasks later in this document).

For the client-server protocol it might make sense to define the protocol in such a way that one may use JSON.parse() and JSON.stringify(). We give some example commands which could be sent to the slaves:

1. to client-1 "flood-screen (255, 0, 0)" to fill the screen with a background color of red;
2. to client-1 "flood-screen (255, 0, 0) 0.2 (0, 255, 0) 0.2 (0, 0, 255) 0.2" to fill the screen with a background color of red for 0.2 seconds, then green for 0.2 seconds and then blue for 0.2 seconds and then restore back to the original background color;
3. to all clients (or to a specific client) "count-down 5 0.4" to show a countdown from 5 to 1 with 0.4 second intervals and then show an explosion or fade the screen to full white and back down;
4. to client-1 "draw-directions 2:94 4:170 exit:190 245" to draw arrows or lines from the (visible) center of the screen in the directions of 94 degrees (with 0 degrees pointing in the up-direction of the screen) and with a label "2", and so on, the labels are optional.

Your team could choose to implement the actual algorithm to detect the slave devices on the server or in the master. However, you should provide a means for the master (or an extra "supervisor"/"developer" device) to issue commands to the slaves, such that you can use the Javascript console of the browser on your computer to issue commands. This guarantees that the development of the actual algorithms (from TC2) and the development of the protocol and client-server behaviour (from TC1) can be done independently.

Things to consider:

1. What happens if you have a new server version and restart the server, but there are still clients which were connected? Consider how to handle this gracefully. Note that there is a version of the application on the server and a version of the client on the devices.
2. What happens when a client crashes? Are resources on the server released? If that client eventually reconnects, is it considered the same client?
3. How much computation time do you give the server for once client? How many clients can connect?
4. What happens if the server crashes or becomes unavailable? Could the client indicate such a "lost connection"?
5. What happens if one of the slaves is a smartphone and receives a phone call? – Also consider this case in combination with the detection algorithm or the applications which will be developed.
6. What happens if text messages pop up on the display of the phone? – Also consider this case in combination with the detection algorithm. Additionally: consider this as a privacy issue!

Resources:

1. Node.js https://nodejs.org/ and https://nodejs.org/api/
2. https://www.tutorialsteacher.com/nodejs/nodejs-tutorials
3. HTML5 living standard https://html.spec.whatwg.org/, in particular read the section "How to read this specification"
4. Websockets https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

5. Socket.io https://socket.io (and in particular the "get started" chat server tutorial)
6. "Socket.io vs websockets" (we tell you to use socket.io at first, however the TC could argue to change technology; but then it should be done early on in the development process)
7. https://www.html5rocks.com/en/tutorials/websockets/basics/
8. Please check the dates of articles and comparisons on the web! Things change!
9. Same-origin policy https://en.wikipedia.org/wiki/Same-origin_policy
10. XSS (cross-site scripting) attacks e.g. https://snyk.io/vuln/npm:socket.io:20120417
11. https://nodejs.org/en/knowledge/HTTP/servers/how-to-create-a-HTTPS-server/

TC2: Image analysis and algorithms

This technical committee will be concerned with solutions for the application logic, developing algorithms to identify the physical slave device configuration, specifying when the algorithms work successfully, and developing strategies to test all this.

The first task is coming up with algorithms to detect slave displays. Note that we are in control of the slave displays, and so it is for example possible to change the color of the slave displays. Two consecutive photos made by the master device could then be used to detect changes in the picture and to discover the positions of the slaves. Sounds easy? It depends on what assumptions you put in. In a first instance think about putting the smartphones from your team on a table and using the master to snap a picture from the top (display screens are neatly parallel to the camera plane). Assume two such pictures from identical positions, one with white backgrounds on the displays and one with black backgrounds. Further assume, completely theoretical of course, that these pictures are pixel-identical except for the color changes in the displays of the slaves. Then, the difference of these two arrays (maybe as grayscale images) will reveal to you where the slave displays are. Further assuming the slaves are not overlapping we can identify blobs of pixels corresponding to each individual slave. Great! Now start relaxing your assumptions to realistic settings...

Understanding what is expected to happen in real life situations will allow to define realistic assumptions. In the extreme case when trying to detect the slave displays in a computer lab we will have to take care of a shaky camera position (the master is hand held by the user) and scaling of objects in the two pictures, changing light conditions, bad light conditions, reflections, shadows, noise in the image, students running through the room, a beamer projecting a movie, one screen partly obscured by another screen, or maybe a person... To develop such an algorithm we will work in stages, going from highly controlled settings into more wild scenarios.

To help you on the way we portray some possible algorithmic suggestions. It is not required that each team implements the same algorithms. On the contrary, we expect there to be quite some different choices.

Things which need to be taken into account are bandwidth and processing power. It is not necessary to send super high resolution images, but of course, we need a slave display to occupy some minimum number of pixels in the images which are input to the algorithm. Higher resolution images lead (hopefully) to higher accuracy in detecting slave displays, but also in higher computational costs and required bandwidth. **As the developer of the algorithm you are making promises about the output given that the input adheres to your "reasonable" and minimal assumptions.** An example of a reasonable assumption is that the person holding the master device should try it's best to take both pictures from exactly the same point of view. Some of these unwanted effects can be detected by the algorithm and a friendly message can then be presented to the user, e.g., "Detection failed, please try to hold your device steady.". **We will test the promises of your algorithms given the assumptions that you specify, and we expect your team to report on repeatable scientific tests which confirm your claims.**

Once blobs of pixels making up one slave device are identified we need an algorithm to detect its orientation and scaling. There are multiple solutions one can think of. In line with the previous algorithm we could snap two more pictures: one with the right half of the screen in the second color and one with the bottom half of the screen in the second color. We could now use the first algorithm again on combinations of these pictures and delivers us four regions for each slave display (NE, SE, SW and NW). Calculating "centers of mass" of these four regions should tell us how the display is orientated on the 2D view from the master device.

Is this the only possibility? No. We could do without taking extra pictures by calculating the "center of mass" of the whole pixel blob and then calculating a 2-by-2 inertia matrix and using its eigenvectors... some hand waving... you should be able to detect orientation and scaling. Which algorithm is better? To decide this we need to **compare assumptions, cost, precision and promises**. That is what we want you to think about!

It should be obvious that the development of these algorithms can happen independent of the application framework. You can load images from disk while implementing and testing the algorithm. You should have easy and hard test cases. Maybe you want to use some test images from other teams as well. Structure your code in such a way that the algorithms are a separate library, and you can run scripts to test your algorithms from the command line. **Testing your algorithms should not involve manually looking at the output.**

Things to think about:

1. If you take a picture of a display with bright red background and then take a second "identical" picture with the display having a black background, how much do you think the other pixels change based on the automatic adjustments your phone's camera makes? (Assuming the captured display takes up a reasonable amount of the picture, the second picture is taken in much darker light conditions, and so the camera will automatically adjust exposure, possibly introducing more noise, but also changing the color of the other pixels.)
2. What input is allowed for each version of the algorithm you develop?
3. What are the minimum requirements?
4. What about reflections in screens? What about shadows?
5. Consider sticking with well defined "easy" algorithms that you can implement yourself and reason about instead of rolling out heavy multipurpose graphics libraries.
6. Take into account that whatever Javascript runs on the Node.js server is single threaded!
7. What precision do you expect from your algorithm?

References:

1. https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia
2. https://mathematica.stackexchange.com/questions/17060/image-processing-finding-orientation-and-position-of-symmetry-axes
3. https://en.wikipedia.org/wiki/Gaussian_blur
4. https://github.com/lovell/sharp/ (simple image manipulation, C++ backend)
5. https://sharp.pixelplumbing.com/en/stable/ (API)
6. https://malcoded.com/posts/nodejs-image-resize-express-sharp/ (example of setting up a (local) module, loading an image from file, manipulating, and serving the manipulated file to the browser – see the docs for saving the result to another file; note that you might need to understand how Javascript uses 'then()' and 'promises')
7. https://github.com/mapbox/pixelmatch

Teams

You are assigned to a team of 6 students. We expect the full team to be present from 14:00 to 17:00 during the P&O sessions. The labs are reserved until 19:00. Each member of a team has to take up a specific responsibility. The team has to decide in the first session which member will take up each task. These tasks are fixed for the full semester.

1. **CEO: Chief executive officer:** The CEO of the team is responsible for making executive decisions. The CEO is responsible for making and maintaining the **project schedule** and making sure everybody is doing their job and delivering on time. The CEO **manages resources** (team members and time) such that the team produces the deliverables before the deadlines. The CEO is the first contact point for the P&O didactic team. The CEO calls for a **team meeting** twice a week and keeps a **ledger** which describes when the meeting was held and who took part (all members should take part). In this meeting the CEO will take note of what each team member is doing and what the team member should do until the next meeting (based on the needs of the project schedule). **The CEO keeps a log of the amount of time the team members spend on the different tasks.** (The team members report this

time to the CEO.) The didactic team might consult the current project schedule and ledger at any time. Therefore, the CEO will post and update them weekly in the master branch of the git repository. Use a folder called **admin** placed in the root folder of the repository and add files **ledger.md**, **schedule.md** and **timelog.md**. The **timelog.md** file should contain the names of the team members and their role in the team (CEO, CTO, member of TC, code reviewer, projects currently working on, ...).

2. **CTO: Chief technology officer:** The CTO of the team oversees all technology decisions in the team. The CTO coordinates which **libraries** are being used, how the **git repository** is organized and how the different team members should use the git system when developing different features in parallel. (See links in Task 1 for more background.) The CTO decides what technology is being used for the team to communicate and share files. The CTO is responsible for **backups**. The CTO puts measures in place to be **able to reproduce tables and graphs which ended up in reports** (this means that the exact version of the code and input should be readily available upon request and there should be a script which can be run and shows that the output is indeed as reported). The CTO makes a **design schematic** and keep this up to date, it should be available in the **admin** directory in the master branch of the git repository. The didactic team might consult the design schematic at any time.
3. Member of TC1: This person will participate in the meetings of TC1 and report back to the team.
4. Member of TC2: This person will participate in the meetings of TC2 and report back to the team.
5. Code reviewer / Security auditor / White hat: The task of a code reviewer is to let others design and implement code and then, when they are finished, or at an appropriate intermediate time, read the source code to check that it properly implements the specified functional and non-functional requirements according to the design, as well as to evaluate the quality of the code and related artifacts. This means that any functional and non-functional requirements analysis/specification and design documents, high-level algorithm descriptions, user documentation, etc. should be available to the code reviewer, as well as any pertinent evidence such as a test suite. The reviewer should point out incomplete specifications and should pay attention to what is considered good software practice and suggest changes. In particular the reviewer could pay extra attention to security risks in the client-server code and implementation errors for the actual algorithms. Note that for security audits, threat/attacker models are a necessary part of the specification; correspondingly, threat analysis/attacker modelling is a necessary part of development. The code reviewer may comment on the quality of these other, non-code artifacts as well. Note also that it is the developer's job to implement a test suite; the auditor's job is to check the test suite's comprehensiveness.
6. Code reviewer / Security auditor / White hat: idem.

Tasks for the first semester

We here list the tasks. The first task should be completed during the first session. Every other task has a "delivery date" on which the task should have been completed and comes with an assignment on toledo which requires the upload of a report on the task. It is expected that tasks will be functional at the delivery date, although maybe not all details have been resolved to a satisfactory level. We will give feedback on the results and reports during the semester.

At the end of the semester we require a full report (incorporating updated and corrected versions of the small reports) and a presentation with demo.

Note that we list delivery dates but we do not state when work on a task could begin. The CEO and the team should decide how to plan the tasks.

Task 1: baby steps during the first session

As a first test run we will ask each member of your team to follow the tutorial of socket.io which creates a chat server. Each member should do this individually. This should get you started with Node.js, HTML5, jquery and socket.io. Don't just copy-paste but try to understand the structure of the code. Note: do not ask for version 4.15.2 of the express package, just ask for the latest package by leaving out the version number. The same

remark holds for including the jquery.js package in the HTML file. Also note that you should install "socket.io" and not "socketio" (which would give you version 1.0.0 instead of 2.3.0).

Next we need you to experiment with incorporating git in your team infrastructure. You will use github.com and put your project in a private repository (using the Github Student Developer Pack this is free of charge). Name your git project **2019-2020-PnOCW-team-#**. Invite the TAs as collaborators to your project (account names: vincentc, timvhamme) during the first session. There are multiple ways of using git in a team. As an experiment we want one member of your team to create a test project and commit and push the initial "Hello world" version of the socket.io tutorial. Then the other team members join in and add the subsequent steps from their own computer (different team members add different things). When you are at the end of the tutorial, and each member pulled the working chat server, try to implement some extra features and see what happens if two team members make changes concurrently. Literally: let two team members implement some extra feature while the others watch, and then observe how they try to merge their changes. Maybe repeat this a few times and work through some merge conflicts. Also let both members change the port to be 80## at the same time with ## your team number. Eventually, the **CTO is responsible for the correct usage and knowledge of git for all the team members**. By the start of the second session the CTO should be fluent in git and should be confident the full team knows what to do. The CTO needs to decide what the "best practices" are for the team (see the two links below). **At the start of the second session all the source should be in git on GitHub.**

At this point the team could be divided into two chunks, one continuing on the client-server architecture (for Task 2) and one starting to understand how to do image manipulation (for Task 3). Both groups will have to understand a bit about how Javascript works as an asynchronous single-threaded language... The first group should try to understand how Node.js handles different clients, the second group should checkout the image resize tutorial in the references of TC2.

https://jameschambers.co/writing/git-team-workflow-cheatsheet/

https://www.atlassian.com/git/tutorials/comparing-workflows

https://www.freecodecamp.org/news/how-to-deal-with-nested-callbacks-and-avoid-callback-hell-1bc8dc4a2012/

https://www.freecodecamp.org/news/javascript-from-callbacks-to-async-await-1cc090ddad99/ (Note: change xhr.response to xhr.responseText)

https://www.tutorialspoint.com/nodejs/nodejs_callbacks_concept.htm

Task 2: basic application framework

Here we make the basic bare bones application framework: the webpage can be served, socket.io is up and running, the master can access the phone's camera and send a picture to the server, the slave nodes can be instructed to change the background color through the Javascript console of a supervisor device and they can be instructed to draw directions from the center of the screen using the same Javascript console.

Deliverable in week 3 (report deadline 14/10 @ 23:59 on toledo).

Task 3: basic slave screen detection

In this task the algorithm(s) for detecting several slave screens lying flat on a table and being pictured from the top are implemented and tested. You are required to specify what you consider reasonable assumptions and minimal assumptions, and how accurate the output will be (depending e.g. on the input resolution or total number of pixels, the shape of each blob, and the amount of pixels per blob). There should be a sufficient amount of test cases which can be run from the command line. You should be able to state what the asymptotic complexity as well as the actual performance of your algorithm is (possibly under certain input regularity assumptions) and verify this by some basic timings. Do not forget that even in this well controlled setup, there might be annoying side effects where the camera adjusts exposure to compensate for the brightness of the displays. Try to keep your algorithm as simple as possible. If you implement multiple algorithms, then please also compare them.

Additionally you need to come up with an algorithm to identify each slave device. The obvious way is of course to light them up one by one, but there might be other algorithms which you could use. Every picture that needs to be taken by the master device will cost time to first set the state on the slaves, then take the picture and transmit it to the server and then run the algorithm.

For every slave device we also need to know its orientation and scaling. Two possible algorithms were already sketched in the description of TC2. Again the same remark holds: fewer pictures might deliver benefits in speed. But this depends on your particular situation, so this needs to be assessed or even tested. (Timings of transferring images and issuing commands could be done using the results from Task 2.)

Stating the properties of your algorithm is required for each algorithm.

Deliverable in week 4 (report deadline 21/10 @ 23:59 on toledo).

Task 4: advanced screen detection

Once the basic screen detection algorithm is working it is time to get your hands dirty. Now you are allowed, actually, required, to pile up the phones on the table such that they may be overlapping and/or their display is not parallel to the picture plane. In the simplest case you could consider the nice configuration of phones on the table and simply tilt the camera. Again you have to think about reasonable and minimal assumptions such that you can still make claims about your algorithm's precision and cost. All algorithms of Task 3 should be adjusted to this use case.

We expect the more chaotic layout of slave screens to be not too hard. Therefore in this task you should try to also take care of more annoying factors such as reflections, irregular sunlight, shadows and other disturbing factors which you might have left out of the reasonable input in Task 3. Your algorithms should also be able to detect unreasonable input whenever possible (and should identify possible cases that might have caused it).

Deliverable in week 6 (report deadline 4/11 @ 23:59 on toledo).

Task 5: triangulate

In this task the application framework and the detection algorithms are going to be integrated. The detection of the slave displays now happens on demand of the master device. Once the physical setup of the slave displays has been computed we want to calculate a triangular mesh between the centers of the screens. Here again you will have to specify what you promise for which reasonable input. A famous algorithm is Delaunay triangulation. Think about border cases: what happens if all centers are on a line? What do you want if they are "almost" on a line? The Wikipedia page on Delaunay triangulation might be a good starting point. Basically we do not want nearly "degenerate triangles". Once the mesh has been calculated we want to see the connection lines on the slave displays. The centers of the screens should be marked with a star. Number the slaves and label the connection lines. The lines and stars should appear to have the same thickness and size across the screens. For configurations as in Task 4 extra care needs to be taken. The CEO could decide to only allow configurations as in Task 3 and leave the configurations as in Task 4 to come into play in Task 7.

Deliverable in week 6 (report deadline 4/11 @ 23:59 on toledo).

Task 6: countdown

This task is about synchronization of the slaves. Put a "countdown" functionality on the master device and test how well the slaves can display the countdown in a synchronized way. You can make a video to analyse how synchronous the slaves act. Obviously synchronizing the clients can be done in multiple ways. We are interested in the countdown being as synchronous as possible. At the end of the countdown the slaves can have some client defined behaviour (i.e., your team can choose how to end with a bang).

Deliverable in week 7 (report deadline 12/11 @ 23:59 on toledo). (Remark this is a Tuesday.)

Task 7: image show off

At this point you should have no problem in figuring out the physical configuration of the slave screens. So let's use them as if they are one big patchy screen, i.e., only where there are slave screens we see part of the big

underlying canvas. The master should now be able to cast a picture (e.g., a predefined set of images, loaded from a URL, or taken by the camera) to the slave screens where each slave screen displays exactly the right patch of the source picture, with the correct transformation. Note that the HTML canvas has a transformation matrix (see the HTML5 link above). Another artefact which might now show is that not every screen might have the same brightness, or the same color temperatures. Bonus points if this is handled in your code.

Deliverable in week 7 (report deadline 12/11 @ 23:59 on toledo). (Remark this is a Tuesday.)

Task 8: video show off

If we can do images we can do video right? Maybe. First it might not be so easy to apply transformations to video, secondly, we might hope that we would be able to start the video in the different slaves at the same time, but it might be tricky to keep them playing synchronously (e.g., by adjusting playback rates).

For this task we need you to scout possible techniques and only if you decide it is possible you should implement this. If you decide this is not possible, then we need to hear a detailed explanation of what the problem is. A logical first thing to check is to start the same video simultaneously on all slave devices without considering them to be one canvas. Does the video stay in sync? How long?

Deliverable in week 8 (report deadline 18/11 @ 23:59 on toledo).

Task 9: animation show off

In this task you will develop a little digital ghost snake which rolls and slides over the table, but is only visible when passing under/over the slave devices. The snake should be long enough to cover the space between slave devices such that we can see it leave and enter at the same time. An easy way to model the moving snake is to use the edges of the triangular mesh from Task 5 and to add some random sinusoidal motion on top of that. The snake could be drawn by simple primitives on an HTML canvas. We want you to add shadow and maybe blurring such that the snake looks alive and three dimensional.

Deliverable in week 9 (report deadline 25/11 @ 23:59 on toledo).

Task X: final semester result

The final semester result should be a demonstration of something lightly amusing like the ***Cat Caster*** from the introduction. The different screens in the computer lab then make a patchy canvas for your cat to explore. You can take the Cat Caster idea and change it in any way you like. For example: cats normally do not take instructions very well, so you will have to make pretty hefty gestures to make the cat jump screens. Or maybe you have to repeat the gesture a couple of times, otherwise the cat will just look annoyed over its shoulder to you. Or it might sneak over to its friend, or run off to a mouse, fall into a bucket or a wormhole. More people can join the game, and you can race each other to the food.

Deliverable in week 10 (report deadline 5/12 @ 23:59 on toledo). (Note the deadline is on a Thursday and involves the final report of the semester, not the report on Task X. Do not forget to take into account the remarks from the code reviewer and the remarks from the didactic team on all the previous reports!)

Reporting

For every task in the above list we will open an assignment on toledo and your team will have to upload a short report (PDF). Depending on how the CEO allocated resources to a task, the work has likely been done by only a subset of the team. The report should mention who did what. The code review and auditing should normally been done by somebody who was not involved in writing the main code. This person should be listed separately. We understand that under time pressure it is not always possible to do a full code review and auditing, nor is it possible to implement the changes suggested by the reviewer. We expect at least one paragraph on the report from the reviewer with remarks and also a statement of how much of the code was reviewed. Similarly, the developer(s) should state if any changes are already implemented or yet to be implemented. For the final report the reviewers comments should be taken into account.

Final report, presentation and demo

The final report is listed as the deliverable for Task X. We will plan presentations and demos in weeks 11 and 12.

Correctly referring to other people's work and "borrowing" images or code

You cannot just "copy-paste" things that you find on the internet or get from friends. Whenever you modify code that you found online, e.g., on Stack Exchange, you are required to put the URL in your code. Whenever you use an image (e.g. in your user interface, or to illustrate your report or your slides) you have to check if the usage is allowed and correctly cite the source. You are required to check software licenses of all the software that you use and make a list. Think about what those licenses imply for your software project and put a license on your project as well.

Especially in your report we expect a list of references. For websites this includes the URL and the date that you visited the website. In your slides we expect to see a correct citation whenever you use an illustration. In your application we expect a credits screen which shows your license and credits for whatever software you are using, or images you are using.