Presentation workshop

Christian Forkstam, PhD student
Karolinska Institute, Stockholm Brain Institute
Donders Centre for Cognitive Neuroimaging and
Max-Planck Institute for Psycholinguistics, the Netherlands
E-mail: **christian@forkstam.se**

Outline

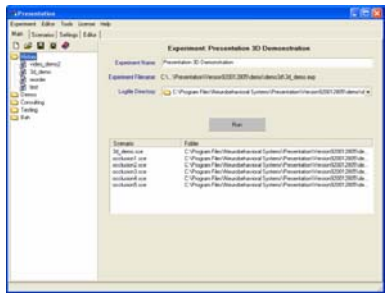
- Presentation software (http://www.neurobs.com/presentation/docs)
- Presentation toolbox: Stimulus Presentation Lab (PCP-SPL; http://forkstam.se/PCP)

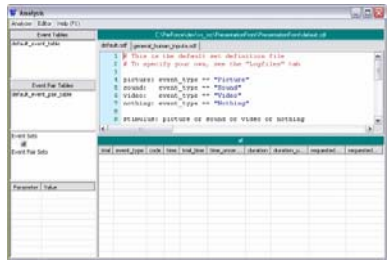
What is Presentation? 1

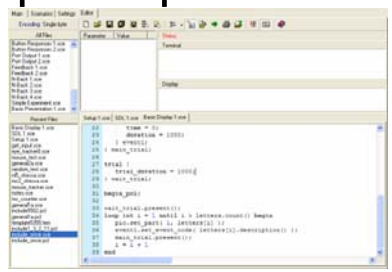
- Presentation is a stimulus delivery and experimental control software for timely stimulus presentation and response logging
- It runs neurobehavioral experiments on Windows using standard PC hardware
- It was designed for behavioural and physiological experiments including FMRI, EEG, and reaction time investigations
- It is completely programmable
- It is also possible to run as a sequence of pages much similar to a animated power-point presentation

What is Presentation? 2

- The Presentation software consists of
  - a Graphical User Interface (GUI)
  - a Scenario Description Language (SDL) which describes stimuli, sequences of stimuli and associated properties, and
  - a Presentation Control Language (PCL) which is a interpreted programming language for custom control of scenarios
  - a text editor designed to facilitate writing and editing scenario files, template files and PCL files
  - a homepage (www.neurobs.com) with a knowledge forum and software documentation with tutorials and example experiments







Features 1

Presentation has many features that make it applicable to a diverse range of applications:

- Multi-tasking
  - Simultaneous control of display of several audio and/or visual stimuli
  - Simultaneous monitoring with high accuracy of responses and I/O ports
  - External software interfacing, e.g. interface to Matlab workspace
- Timing
  - Reporting of times for any event of interest and verification of all timing to detect operating system problems
  - Benchmark testing to evaluate operating system timing behavior and to test individual computers with individual experiments
- Programmability
  - Built-in programming language for custom control (PCL) to control stimulus presentation in real-time
  - Adjustment of stimuli based on responses or external inputs in real-time to use e.g. adaptive procedures during presentation
- Visual Stimuli
  - Display of stored images, stored videos, animation, and 3D stimuli
  - Synchronization of stimulus appearance and display of a new stimulus up to every vertical scan

Features 2

- Auditory Stimuli
  - 1-8 channels of stored sounds (.wav) with a resolution of 8, 16, or 24 bit
  - Independent volume control on each channel
- Input Devices
  - Monitor button press input from e.g. mouse and keyboard
  - Monitor position input from mouse and joystick
- Input/Output ports
  - Simultaneous monitoring of multiple input and output ports
  - Constructs an interface to external devices and acquisition systems by the sending of event time-locked pulses
- FMRI (external device) synchronization
  - Synchronization of stimulus sequences on specific pulse/scan (use of interrupts with the parallel port)
  - Option to test FMRI experiments in emulation mode
- Eye-tracker interfacing
  - Generalized common interface for communication with eye trackers
  - Access gaze position, blink, fixation, pupil, and saccade data
  - Present eye position dependent stimuli

System requirements

- Any Microsoft Windows operating system (except Windows 95)
- DirectX version 8 or later (a software library available for free from Microsoft)
- A Pentium class or compatible processor
- Hardware needed to run your experiment:
  - Is the hardware capable of presenting the stimuli in the desired manner?
  - Can the hardware achieve the required timing accuracy?
- Important characteristics of your system are the amount of memory, the sound card and the video card
- Windows will swap to the hard drive if the memory is insufficient
- This results in disk accesses occurring during the experiment which will greatly degrade performance
- If the stimuli for one scenario do not all fit into memory simultaneously, you can load and unload stimuli during the scenario

Hardware configuration

- The GUI allows you to set the experimental hardware options

Example:

The experiment description specifies the visual stimuli for an experiment.

It assumes the presence of four buttons to gather subject responses.

We then use the GUI to select the display adapter (and therefore the monitor) to use for stimulus presentation.

We also specifies the response devices and specific buttons to use.

Stimuli

- Presentation can display 2D graphics, compressed videos, sounds, 3D graphics, and force feedback
- Presentation can construct text displays and simple 3D shapes for you
- More complex stimuli must be constructed outside of Presentation and stored on disk for use in the experiment

Timing 1

- Presentation is designed to attempt to present the stimuli exactly as requested
- More importantly it produce a detailed report indicating what actually happened during an experiment
- Both performance and accuracy is required for good timing control
- The concern of accuracy is always relevant to the concern of exact timed presentation
- Without knowing the timing accurately, it is not possible to verify that the stimuli were presented as desired
- Factors affecting the ability to determine the actual timing of events:
  - The hardware involved
  - Communication method with the PC and device driver and system handling
  - Activity of the software when an event occurs
  - Activity of the operating system or other programs (including Presentation) when an event occurs

Timing 2

- Presentation has special features to address the accurate measurement and verification of times measured by the software
  - First, to have accurate information about event timing one must know something about the hardware involved
  - Software can usually report the time of detection by the software accurately
  - **But the relationship between that time and the time for some physical event is determined by the particular hardware involved**

Example: A response device scans its button state every 10 millisecond. This makes it possible to know the time of a response with at the most a 10 millisecond resolution. This is a hardware property and is independent of any software level inferences in the rest of the system. It places a hard constraint on the accuracy possible using any software.

Timing 3

- Second, information from hardware is transmitted to application software using e.g. a USB or PCI bus, and is then handled by device drivers within the operating system
- Each with its own mode of operation and associated timing behavior which often is not directly controllable by software applications
- **If an event occurs while the software is performing an operation, the event cannot be detected until the operation is completed and the software can again check for events**

Timing 4

- Windows processor time slices are on the order of tens of milliseconds
- But Presentation performs many operations at once (playing sounds, preparing stimuli, monitoring ports, etc.)
- For this Presentation use very small micro-threaded operations to minimize the time between checks for events
- Under most circumstances, these time intervals are well below 1 millisecond
- Interrupts by the operating system, device drivers, or other applications is a big problem for accurate event detection
- Operating system interruptions cannot be predicted or prevented
- In most cases these interruptions will not adversely affect the timing accuracy of the experiment
- But it is important to verify the timings to avoid unexpected events cause timing problems without notification to the experimenter

Timing 5

- Presentation verifies the events times it measures using its uncertainties feature
- For all events that can be affected by operating system activity or Presentation's own activity, Presentation always measures two times
- One of which is definitely before the detection of an event by the software, and one of which is after the detection
  - Reporting times for any event of interest to the user allowing to verify that the stimulus sequence was presented exactly as intended
  - Reporting timing uncertainties for all events to notify the user of any interruptions in the stimulus delivery or response monitoring
- This ensures that no such interruptions can affect your timing without the knowledge of the experimenter
- This allows for the monitoring responses while presenting both complex and high rate multimodal stimuli without sacrificing response timing validation

Timing - Summary

- For most systems and experiments, Presentation attains a precision of better than one millisecond

- To achieve the best possible timing
  - Carefully control the software running on the Presentation system
  - Minimize any background processes
  - Turn off scheduled task
  - Turn off unnecessary running applications
  - Turn off the network to avoid any network related activity
  - Maintain your experimental computer installation as clean and minimal as possible

Timing and uncertainties for events 1

- Presentation is designed to receive explicit notification if any time uncertainties occur
- For each event in the log file Presentation provides a time of occurrence T and an uncertainty dT
- These two numbers provide bounds on the time of occurrence of a Presentation event
- The relationship between Presentation events and actual events depends on the particular type of event and the hardware involved
- The uncertainty value means that the Presentation event occurred between time T and time T + dT
- For each type of event, Presentation determines if the event has occurred (**is_done()**) which returns true if the event has occurred and otherwise false

Timing and uncertainties for events 2

- At some point near the occurrence of an event, the following sequence occurs in the program:

```
...
time = time1
is_done() is false
...
time = time2
is_done() is true
[start of output code]
time = time3
...
```

- This means that the Presentation event occurred some time between the first and second calls that are shown to is_done()
- The event occurred after time time1 and prior to time time3
- Presentation reports the following T and dT values in the event log

```
T  = time1
dT = time3 - time1
```

- This timing cycle is performed for every event

Timing and uncertainties for events 3

- Any interruptions occurring during the experiment which affect the reported times of events will be in the reported uncertainties
- Presentation allows you to send output codes associated with events from the standard serial and parallel ports
- The beginning of these codes is also bounded between times T and T+dT
- This means that the start of an output code and the associated Presentation event are never separated by more than dT

Presentation programming 1

- Presentation uses a simple text description both to describe the stimuli to present during an experiment and how to present them
- The text file experiment specification describes stimuli, sequences of stimuli and general experimental behaviours
- An experiment implemented in Presentation will consist of one or more scenarios
- The chain of events in an experiment is specified in scenarios by writing a text description stored in one or more text files for each scenario (SCE, PCL, TEM)

Presentation programming 2

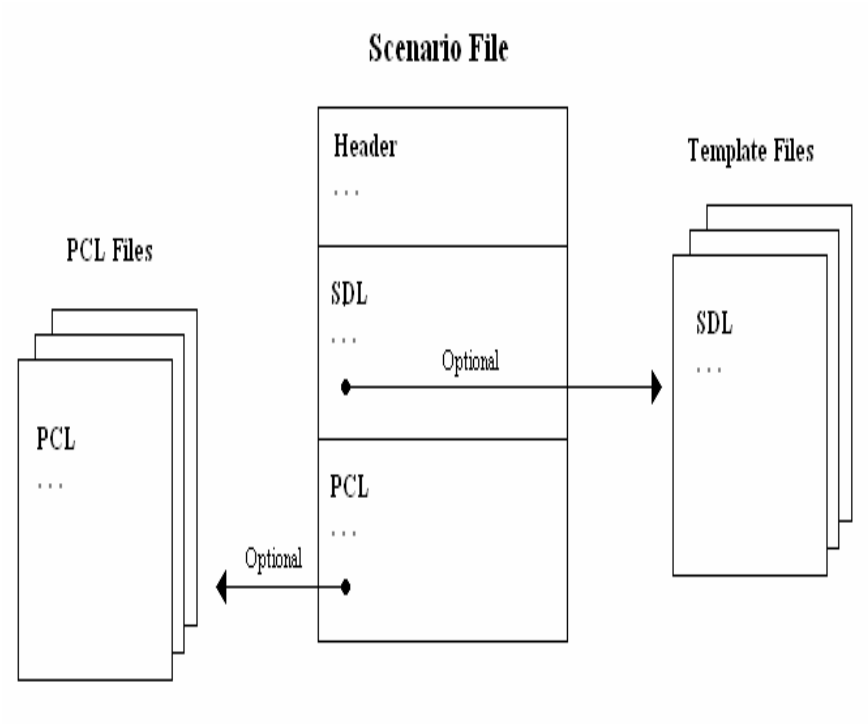
- Scenarios are the core unit in Presentation and experiments are subdivided into scenarios
- A scenario is a sequence of actions that Presentation performs continuously before returning control of the computer back to the experimenter
- When a scenario starts, Presentation takes control of the computer's resources and executes the actions that you have specified in the scenario description
- You cannot regain control of the computer until the scenario ends, but you can pause, resume, or quit the scenario at any time using pre-specified keyboard keys
- You control the subdivision of an experiment into scenarios
- A scenario may contain one block or many, one test or many tests, one experimental condition or many

Presentation programming 3

- There are two custom languages that are used to write scenarios
  - Scenario Description Language (SDL): descriptive language used to specify stimuli and their associated properties, and sequences of stimuli and associated properties
  - Presentation Control Language (PCL): interpreted programming language used to implement custom control of scenarios
- All scenarios will contain some SDL, and some are written entirely in SDL
- For each scenario, there is one main text file called the scenario file, which identifies the scenario
- When you specify a list of scenarios when making a Presentation experiment, you do so by specifying the filenames of the scenario files

Presentation programming 4

- The text description of a scenario has three distinct parts.
  - The first part is called the **scenario file header**, and can contain definitions of various parameters that affect the scenario as a whole
  - The second part is the **SDL object definition** part, which described the stimuli and stimulus sequences that will be used in the scenario
  - The third part is the PCL program, which is optional and can be located in a separate PCL text file



Presentation programming 5

- It may be convenient to separate the scenario description for a single scenario into multiple files
  - First, you can place sections of SDL into separate template (TEM) files that are referenced by the SDL section of the scenario file
  - The text of these helper files is imported by Presentation when it analyzes the scenario file
  - These template files may in turn reference other template files
  - Second, instead of putting the PCL program inside the scenario file, you can place it in a separate PCL text files
  - Also the PCL files may reference other PCL files that are imported when Presentation analyzes the PCL program

Scenario Objects 1

- After the scenario header a series of statements that define the various components of the stimuli to be used by the scenario is defined
- The SDL section is constructed using scenario objects, an abstraction within Presentation that represents some aspect of the stimulus delivery
- The syntax of SDL allows you to specify your scenario objects
- Examples of scenario objects are **picture**, **bitmap**, **sound**, **trial**, and **stimulus_event** objects
- Scenario objects have associated parameters and can refer to other scenario objects
- For example, a picture object, which represents one full screen of graphics, can contain one or more picture part objects, such as bitmap or text

Scenario Objects 2

- Scenario object definitions have a specific format
- The definition of a scenario object begins with the name of the type of object

```
#-- SDL excerpt --#
# In this example, picture, bitmap, and trial are object type names
picture {} default;
bitmap { filename = "pic1.jpg"; } graphic1;
trial {
   picture pic1;
   time = 0;
};
```

- Every time you write an object definition, a new object will be created, even if that object is identical to some other object

Scenario Objects 3

- After the type name, an object definition always requires an opening curly bracket **{** followed by one or more statements, followed by a closing curly bracket **}**
- You can then optionally assign a **name** to the object to store the object instead of immediate use
- An object definition, like all SDL statements, always ends with a semi-colon **;**

SDL Type Definition: scenario object

<keyword *object_type_name*> {
    [statements]
} <string *object_name*>;

SDL Versus PCL

- A text description of a scenario is as a recipe that Presentation uses to cook up your experiment
- A recipe usually has a list of **ingredients**, and a set of **instructions**
- The list of ingredients lists things to use when cooking
- The list of instructions tells you what to do with those things

Example: Cooking Experiment

Ingredients (SDL)
2 cloves garlic
1 tbsp salt
½ tsp black pepper
1 cup flower
2 large eggs
…
Instructions (PCL)
1. Mix together eggs and flower in a big mixing bowl
2. Saute onions and garlic in a small sauté pan
3. …

SDL is not a programming language 1

- SDL is as an easy way to specify the scenario objects that will be used during the scenario
- SDL statements are not commands or instructions that Presentation follows when running the scenario
- The SDL part of a scenario is processed before the scenario even starts
- The ingredients for the scenario are
  - a set of scenario objects whose initial state is specified in SDL
  - a trial order list for the trial objects defined in SDL

SDL is not a programming language 2

- After processing the SDL part of your description, Presentation will make all of the objects that it needs
- A trial object contains a list of stimuli and associated timing parameters
- When Presentation presents a trial, it presents the stimulus sequences described in the trial
- The display of trials has some feedback capabilities but for the most part you cannot implement response dependent behaviour in SDL
- If there is no PCL program to follow, Presentation will use the trial order list and automatically present each trial object

SDL automation features

- SDL features looks like programming structures but they are simply text replacement devices
  - SDL Variables
  - Templates
  - Mathematical Expressions
  - LOOPs
  - Ifs
- They are not instructions that Presentation follows when the scenario is run
- They are eliminated by a pre-processor before Presentation ever really sees them
- For programmatic behaviour PCL has to be used

Template Files

- Template files supplement scenario files, and help make writing a scenario easier
- Like scenario files, template files are text files and are written in SDL
- Template files are referenced within a scenario file using a TEMPLATE statement
- This causes Presentation to process the code in the template file a multitude of times
- The use of template files is merely a convenience and are useful for repeated structures
- By the time the part of Presentation that analyzes the SDL sees the description, the code from different files has already been merged making it irrelevant from which file it came
- Template files are often used in combination with SDL variables
- This allow the resulting code to be different each time the code in the template file is processed
- This is useful when you need to repeat essentially the same section of SDL with minor modifications

Template example

```
#-- SDL excerpt --#
trial {
   picture default;
   time = 0;
   picture pic4;
   time = 1000;
   code = "tree";
   target_button = 1;
};
trial {
   picture default;
   time = 0;
   picture pic2;
   time = 800;
   code = "horse";
   target_button = 3;
};
trial {
   picture default;
   time = 0;
   picture pic6;
   time = 1200;
   code = "rose";
   target_button = 1;
};
```

It is irrelevant to Presentation which of these ways is chosen to code

```
#-- SDL excerpt --#
TEMPLATE "main.tem" {
   pic   ecode    isi      tbutton;
   pic4  "tree"   1000     1;
   pic2  "horse"  800      3;
   pic6  "rose"   1200     1;
   ...
};

#-- template file excerpt--#
# filename: main.tem
trial {
   picture default;
   time = 0;
   picture $pic;
   time = $isi;
   target_button = $tbutton;
   code = $ecode;
};
```

PCL is a programming language 1

- Understanding how to use PCL introduces a level of complexity in implementing experiments
- It provides you with a greater level of flexibility
- In scenarios that do not use PCL, Presentation will automatically execute all of the trial objects described in SDL
- For scenarios that use PCL, Presentation executes the PCL program
- SDL is then used to describe the stimulus objects used in the scenario while these objects are then manipulated using PCL

PCL is a programming language 2

- The Presentation Control Language (PCL) is a simple interpreted programming language
- Although PCL is an interpreted language programs are pre-compiled into an intermediate form making them very fast
- The PCL interpreter is integrated into Presentation so that Presentation continues to monitor multiple devices with sub-millisecond precision
- In addition to the stimulus objects described in SDL, the PCL program has access to other run-time information provided by Presentation
- PCL provides tools that permit experiments to be programmed in a compact, efficient manner, e.g. randomized stimulus variation tools
- PCL is a real programming language and can control almost any kind of experimental behaviour

PCL is a programming language 3

```
# Example: The PCL program placed in a separate file
...
pcl_file = "my_pcl_program.pcl";  # define the scenario file in the header
...
begin;
...

# Example: Placed inside the scenario file

# SDL header
...
begin;
# SDL part
...
begin_pcl;
# PCL part
...
```

Scenario Object Access

- SDL allows you to define various objects related to stimulus delivery, for example pictures, trials, and bitmaps
- In SDL you can assign names to any of these objects
- In scenarios that use PCL, the naming of objects in SDL serves the purpose to allow for manipulation of these objects from PCL
- PCL has a set of variable types called reference types
- The value of a reference variable is a kind of pointer that references an object
- When you name an object in SDL, Presentation automatically creates a reference variable in PCL that points to that object
- You can use these variables to manipulate the referenced object

More Presentation programming 1

- SDL and PCL can contain user comments and spaces which are ignored by Presentation
- Comments are notes that programmers make to themselves and other programmers, and can greatly assist editing and debugging stages
- With an **include** statement PCL code from a separate file is read
- The PCL **loop** construct allows you to create PCL code that will execute a single section of code repeatedly

```
loop
   statement_list
until
   boolean_expression
begin
   statement_list
end
```

More Presentation programming 2

- The PCL **if-then** construct allows you to build conditional statements

```
if
   boolean_expression
then
   statement_list

elseif
   boolean_expression
then
   statement_list

else
   statement_list
end
```

- All statements contained in a loop statement between loop and end form a **scope**
- Variables declared in a statement inside the loop statement have no meaning outside the scope

More Presentation programming 3

- **Subroutines** converts lines of code into logical subgroups that allow for a greater flexibility and easier maintenance
- Properly implemented, subroutines provide a modular structure to PCL code
- You may define your own subroutines that can be called later in your PCL program much similar to other objects
- You may pass arguments and receive return values from subroutines
- A subroutine is defined in a subroutine statement (optional parts inside []):

```
sub
   [ return_value ] name [ argument_list ]
begin
   subroutine_body
end
```

PCP Presentation Toolbox

- PCP-SPL (Pipeline Code Project - Stimulus Presentation Lab) is a package for timely stimulus presentation and response logging
- It is an open source project, provided 'as is' and available to the scientific community as free but copyright software
- PCP-SPL runs on the Presentation Software (www.neurobs.com)

TRIAL STRUCTURE: Sub-trial time periods within a trial

Trial name	Screen	Duration (ms)	Time (ms)
BEGIN TRIAL			0
PRE-STIMULUS PERIOD	+	1000	1000
STIMULUS PERIOD	XYZ	3000	4000
POST-STIMULUS PERIOD	+	1000	5000
RESPONSE	Response	2000	7000
INTER-TRIAL PERIOD	+	4000	11000
NEXT TRIAL			

Template files

- PCP-Scenario-Template.sce
  - Scenario header definitions
  - Scenario object definitions
- PCP-Items-Template.pcl
- PCP-Header-Template.pcl
  - PCL header definitions
  - Experiment lego
- PCP function library