Prospettive Informatiche – LM18

Incontro con **prof.Giovanni Gallo**
*(docente dell'insegnamento **'Advanced Computer Graphics'**)*

giovanni.gallo@unict.it

Di cosa parleremo:

- *‘Advanced Computer Graphics”,* cosa si studia?;
- Visual programming: panoramica;
- Visual programming per la Computer Grafica:
  - Compositing
  - Geometry nodes: esempi

‘Advanced Computer Graphics”, cosa si studia?

Insegnamento “recente” attivato dal 2023-24.
Numero di crediti: 6
Ore di lezione: 48

Motivazioni

- Coprire argomenti di Computer Grafica che non si son potuti trattare in Computer Grafica triennale: ma quali?
- Introdurre gli studenti di **Informatica** a tecniche **informatiche** avanzate utili anche oltre la grafica (non sono studenti di Belle Arti!);
- “Educare” gli studenti ai temi della visualizzazione tecnico-scientifica e alla comunicazione mediante grafici;
- Mostrare pregi e difetti del Visual Programming con esempi in Computer Grafica.

Argomenti scelti per l'insegnamento nell'AA 25-26

Parte 1
- Leggi della percezione visiva
- Principi di base del graphic design
- Laboratorio con le librerie *Matplotlib, Pandas, Seaborn*;
- "*Grammar of graphics*": approccio razionale alla visualizzazione dati;
- Laboratorio con la libreria *Plotnine*;

Parte 2
- Richiami di uso del software *Blender*
- Programmazione visuale nel post-processing di immagini CG;
- Programmazione visuale nella creazione di *asset parametrici*: i *geonodes* di Blender

Next year? Qualcosa sarà aggiornata in base alla esperienza dell'anno corrente.

Panoramica sul Visual programming

La **programmazione visuale** rappresenta un approccio innovativo alla creazione di software che sostituisce il codice tradizionale con elementi grafici intuitivi.

L'idea fondamentale è sfruttare la capacità umana di elaborare meglio le informazioni quando sono presentate in forma visiva. Attraverso l'utilizzo di *icone, diagrammi e interfacce grafiche*, la programmazione visuale semplifica la comprensione e la manipolazione dei processi logici sottostanti.

I linguaggi di programmazione visuale sono progettati per essere accessibili a tutti, anche a chi non ha esperienza pregressa nella programmazione, grazie alla loro interfaccia intuitiva e all'utilizzo di elementi grafici facili da comprendere.

Storia della programmazione visuale (1)

Il concetto di programmazione visuale affonda le sue radici negli anni '70, quando ricercatori iniziarono a riconoscere la maggiore efficacia dell'apprendimento e della comprensione attraverso elementi visivi rispetto al codice testuale.

Tuttavia, lo sviluppo e la diffusione di questi sistemi sono stati ostacolati dalle limitate risorse hardware e software disponibili all'epoca.

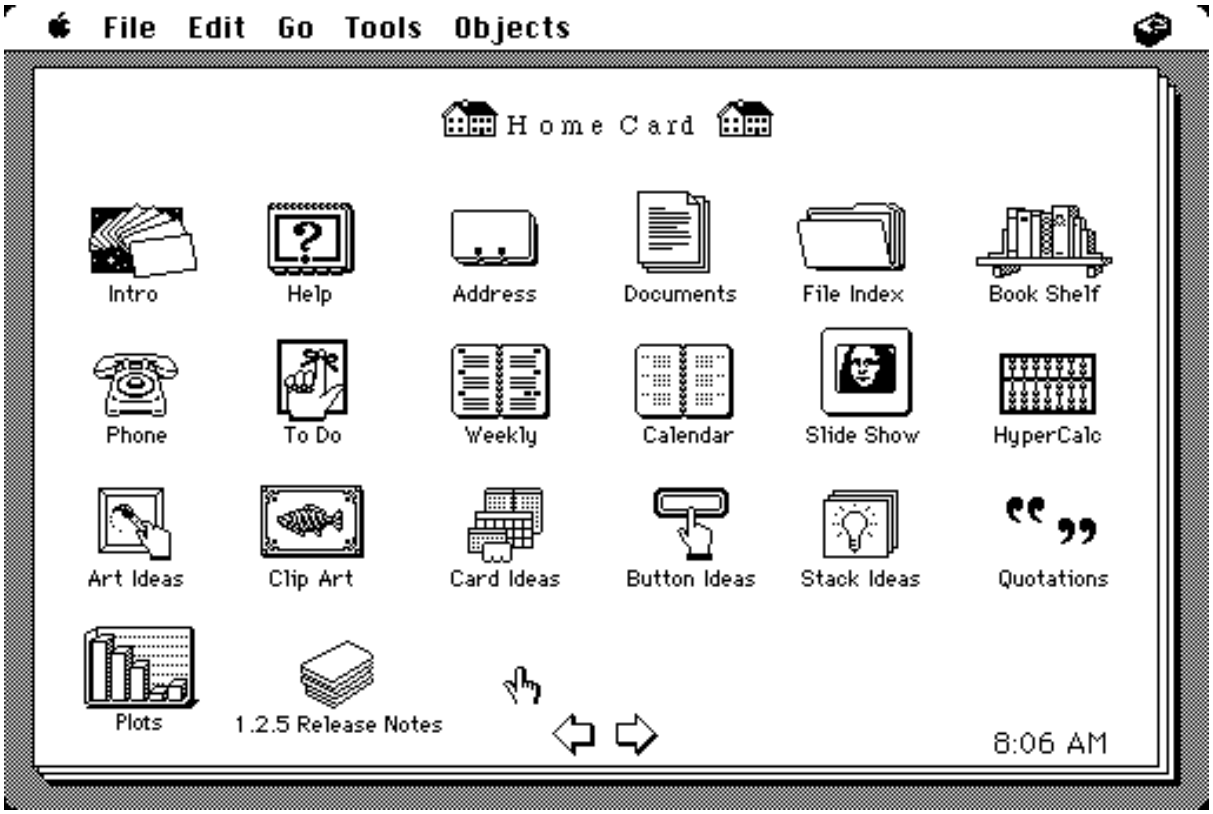
Un sistema precursore dei moderni ambienti di sviluppo visuale per pagine e applicazioni virtuali fu **HyperCard**, disponibile sui primi sistemi Apple nel 1987 con il System 6..
HyperCard offriva un approccio grafico alla gestione delle risorse, in contrasto con l'approccio basato su comandi testuali della "shell" Unix o MS-DOS, che richiedeva una conoscenza approfondita dei comandi da inserire manualmente.
Fu ritirato dalle vendite nel marzo del 2004.

HyperCard non solo democratizzò la creazione di software, ma ispirò anche lo sviluppo di numerosi altri sistemi di programmazione visuale.

Hypercard fu in anticipo sull' HTML, ed è stato uno dei primi sistemi per creare ipertesti.
Cunnigham lo usò nel programmare un abbozzo di ciò che avrebbe sviluppato successivamente sotto il nome di **wiki**.

Storia della programmazione visuale (2)

Hypercard era più un DB per informazioni testuali e grafiche che un linguaggio anche se utilizzava un linguaggio proprietario l'**Hpyertalk** che ne estendeva in qualche modo le funzionalità.



Storia della programmazione visuale (3)

L'avvento del World Wide Web negli anni '90 diede una forte spinta alla programmazione visuale. Sistemi come **Visual Basic**, sviluppati da Microsoft, divennero popolari per la creazione rapida di applicazioni Windows grazie al loro ambiente di sviluppo intuitivo e all'uso di componenti predefiniti. *Anche se di 'visual' non aveva tantissimo!*

Allo stesso tempo, emersero piattaforme online che offrivano interfacce drag-and-drop per creare siti web e applicazioni senza scrivere codice (anche se spesso con limitazioni).

Dopo un lungo periodo di 'stasi' nel settore (con alcuni notevoli sviluppi settoriali, vedi seguito) la programmazione visuale ha subito un'ulteriore evoluzione con l'ascesa delle piattaforme *"low-code" e "no-code".*

L'intelligenza artificiale sta giocando un ruolo nella programmazione visuale, con strumenti che possono generare automaticamente codice a partire da modelli grafici o descrizioni in linguaggio naturale.
Certamente la integrazione di queste due metodologie porterà a sviluppi estremamente interessanti e avanzati.

Storia della programmazione visuale (4)

La *'vocazione settoriale'* è oggi un elemento universalmente presente in tutti i sistemi di visual programming in uso.La programmazione visuale è stata utilizzata in campi scientifici quali:

- **ingegneria:**
  - **VIEW**, (aka ***labVIEW***) acquisizione dati, controllo strumenti, automazione industriale.
  - **Arduino IDE**
- **matematica:**
  - **Geogebra**, per esplorare concetti di geometria nel piano e nello spazio
- **data analysis e machine learning :**
  - **KNIME**
- **digital game development:**
  - **Unity's Bolt,**
  - **Unreal Engine's Blueprints,**
  - **Godot Engine's Visual Scripting,**
  - **GameMaker Studio 2,**
  - **Construct 3.**
- **computer grafica**
  - **Nuke**
  - **Blender**
  - **Da vinci**
  - **Houdini**

Linguaggi di programmazione visuale generalisti

Si citano i più famosi e diffusi

Scratch (sviluppato dal Lifelong Kindergarten Group del MIT Media Lab nel 2003-2007) specializzato per introdurre i bambini alla programmazione.

Blockly (sviluppato da Google nel 2011-12)

Programmazione Visuale vs. Tradizionale: Un Confronto

PRO: La programmazione visuale si fonda su alcuni principi fondamentali della buona progettazione software:

- **Riutilizzo:** i blocchi di codice sono riutilizzati in diverse parti del programma semplificando lo sviluppo.
- **Modularizzazione:** il programma viene strutturato in moduli autonomi, facilitando la comprensione (ma attenzione!).
- **Information Hiding:** i dettagli di implementazione sono nascosti dietro interfacce chiare, proteggendo l'utente dalla complessità interna del codice.

CONTRO: Relativamente alla **correttezza** e al **debugging** la programmazione visuale è *vincente solo in parte*.

I sistemi più avanzati guidano l'utente riducendo la probabilità di errori.
Questo si traduce spesso in programmi più robusti fin dal primo sviluppo.

La manutenzione è un aspetto più incerto. É facile incorrere in architetture "spaghetti" (complesse e difficili da seguire) se il progetto non viene strutturato con cura.
In questi casi sorgono problemi di leggibilità e difficoltà nella comprensione e modifica da parte di sviluppatori esterni o meno familiari con il progetto originale.

Ma ci sono dei limiti….

Problemi di scalabilità

Con l'aumentare delle dimensioni della base di codice è difficile mantenere e adattarsi a un approccio di programmazione visuale: la programmazione visuale potrebbe non essere adatta per progetti di sviluppo software su larga scala.

Con l'aumentare della complessità del programma, i linguaggi di programmazione visuale possono diventare difficili da gestire e da sottoporre a debug.

Prestazioni

I linguaggi di programmazione visuale potrebbero produrre applicazioni meno efficienti di un'applicazione programmata in modo tradizionale.
La programmazione visuale richiede più risorse di memoria e potenza di elaborazione.

Curva di apprendimento per programmatori esperti

La programmazione visuale differisce dalla programmazione tradizionale.
I programmatori esperti debbono effettuare un cambio di mentalità e abitudini.

Modellazione procedurale (da Wikipedia)

https://en.wikipedia.org/wiki/Procedural_modeling#References

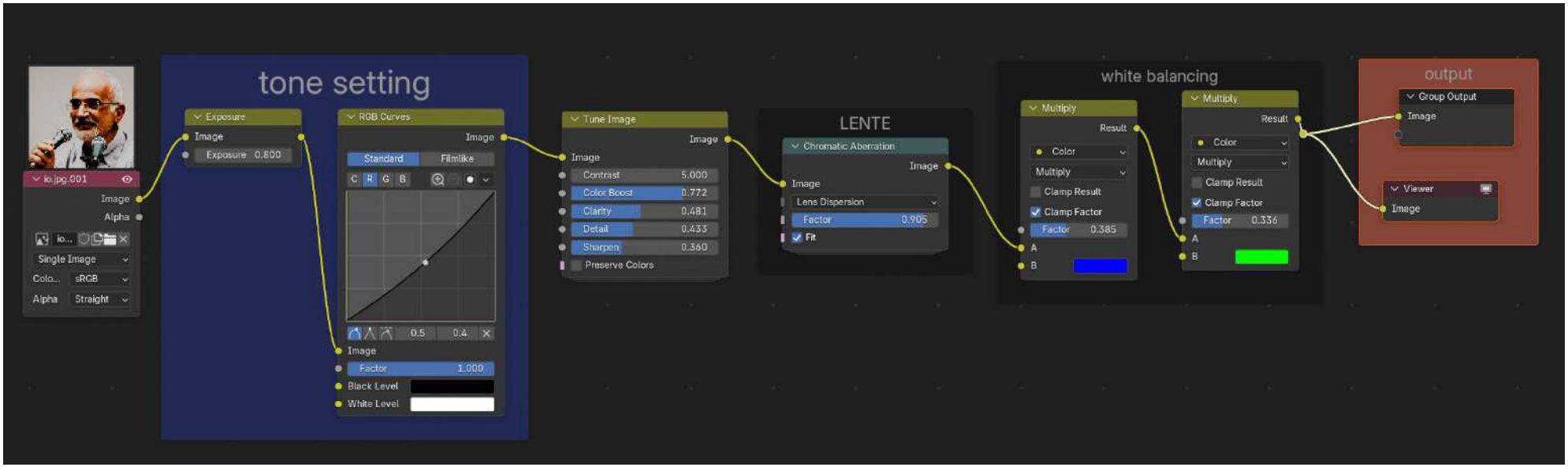
La modellazione procedurale è un termine generico per una serie di tecniche di Computer Grafica che consentono di creare modelli 3D e texture a partire da insiemi di regole facilmente modificabili nel tempo.
L-systems, frattali e modellazione generativa sono tecniche di modellazione procedurale poiché applicano algoritmi per la produzione di scene
L'insieme di regole può essere incorporato nell'algoritmo, configurabile tramite parametri, oppure separato dal motore di valutazione.
L'output è chiamato contenuto procedurale e può essere utilizzato in videogiochi, film, caricato su Internet oppure modificato manualmente dall'utente.
I modelli procedurali spesso presentano un'amplificazione del database, il che significa che scene di grandi dimensioni possono essere generate da un numero molto inferiore di regole.
Se l'algoritmo utilizzato produce sempre lo stesso output, non è necessario memorizzarlo. Spesso, è sufficiente avviare l'algoritmo con lo stesso seme casuale per ottenere questo risultato.

La modellazione procedurale viene spesso applicata quando sarebbe troppo complicato creare un modello 3D utilizzando modellatori 3D generici o quando sono necessari strumenti più specializzati. Questo è spesso il caso di piante, architettura o paesaggi

BLA BLA BLA…. Meglio fare degli esempi?

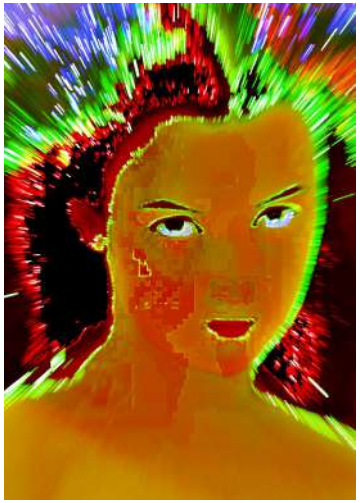
Visual scripting per elaborare immagini (Compositing)

Esempio. Eseguire una sequenza di operazioni su una immagine adottando ad ogni passo parametri opportuni.

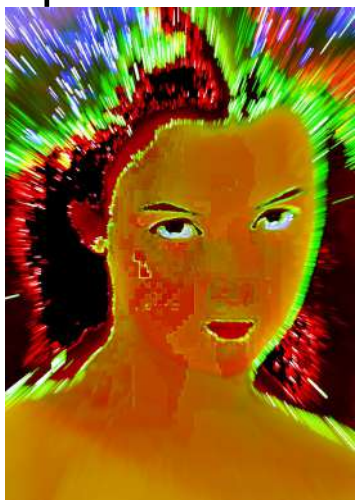


Una sequenza di ‘nodi’ (ciascuno rappresenta una elaborazione secondo assegnati parametri) codifica un ‘workflow’.

Rende facile riprodurre un effetto su differenti immagini



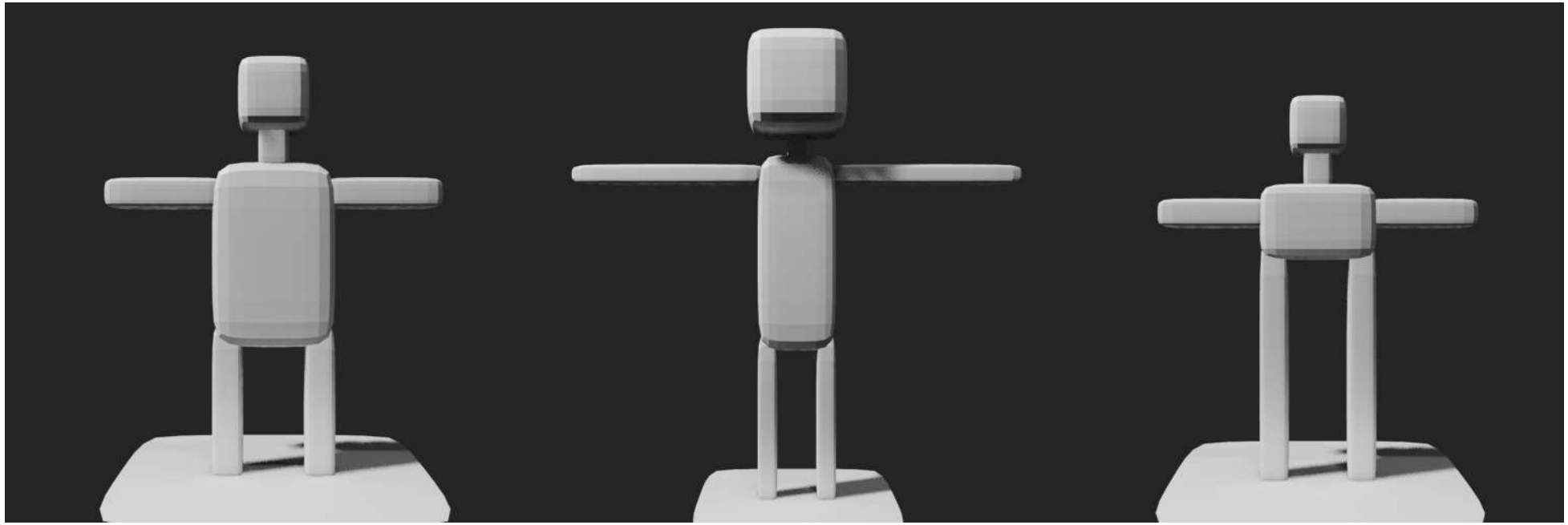
Rende facile modificare il workflow per adattarlo a diverse esigenze



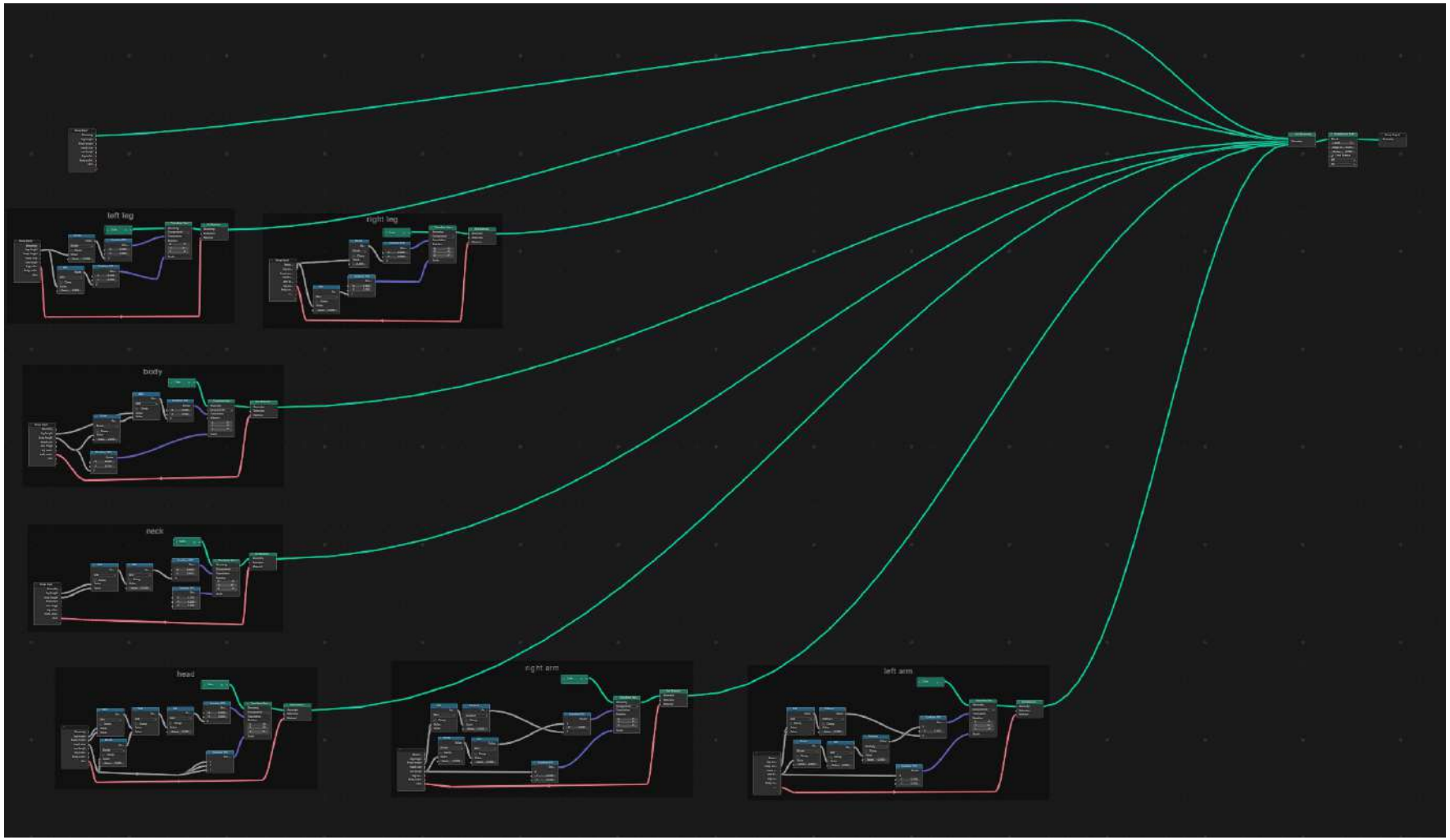
Visual programming per creare asset

“Riutilizzo” e **“Varietà”** sono due qualità nello sviluppo di asset in CG. La ***modellazione parametrica*** è un metodo che riesce a garantirle entrambe. Anche in questo caso la programmazione ‘a nodi’ assiste.



Demo time!

Parametric puppet! (vedi video)



Demo time!

Parametric landscape